1　稳定成熟的技术还是未成熟的新技术?

第 1 章 选择市场

你马上就要进行一次大的投资，也许并不是要投入大笔金钱，而是时间，是你的一生。大都数人对待工作的态度往往都是顺其自然，走一步看一步--我们刚刚深入了解了 Java 或者 VB，老板有一天突然参加了一个热门技术的培训，于是我们就转而学习新技术，直到有人又把新的东西递到我们手里。我们的职业道路就是由一连串没有方向的偶然构成的。

在《程序员修炼之道》一书中，Dave Thomas 和 Andy Hunt 谈到了编程中的偶然性。下面这个场景，会引起大部分编程员的共鸣：当你开始做一个程序的时候，或许手头上有一个从网上复制的示例程序，看上去这个程序可以使用。为了满足你的需要，你会对这个程序稍加改动--添加一些代码，再加一点。你根本就不知道自己在做什么，只是不断地做一些小的修改，直到这个程序完全满足你的需要。但问题是，这样做就像是用纸牌搭建房子，每增添一张纸牌，就增加了一分纸房子坍塌的危险。你根本就不知道这个程序是如何工作的，所以你每做一点儿改动，都有可能导致你的程序完全失败。

作为软件开发人员，用这种投机取巧的方式来编程显然不是什么好主意。但是很多人正是让偶然来决定职业道路上的各种选择。我们应该在哪种技术上投资？应该专注于哪个领域？是应该扩展知识面，还是深入学习一门学问？这些问题都是值得我们细细斟酌的。

想象一下你开了一家公司，现在正要生产你们的明星产品。如果这个产品失败了，公司就会破产。你会花多少精力来思考此产品的消费者是谁？在产品进入生产流程之前，你又会用多少时间来弄明白这个产品到底是什么？我相信你肯定会仔仔细细地考虑其中的每个小细节，然后亲自做出决定。

但是，在职业道路上，面临选择的时候，我们为什么就缺少了这番心思呢？如果你把自己的职业当成是一门生意（事实上它就是一门生意），那么你的"产品"就是由你提供的服务构成的。这些服务是什么？你又会把它们出售给谁？接下来的一年，对此种商品的需求是会增加还是减少呢？在这些选择上你愿意投下多少赌注？

读完本章的内容后，你会找到答案。

1　稳定成熟的技术还是未成熟的新技术？

如果你想投资，可以有许多方法。你可以把钱存进银行，但是利息的增长往往跟不上通货膨胀的速度。买国债也是个办法，但同样，收益也不会很高。不过，这两种投资方法都无需承担什么风险。

你也可以选择把钱投入一个小规模的创业公司。投入几千美金换取公司的一小部分股份。如果公司的决策正确，而且这个决策被有效地执行了，那么你就有可能挣一大笔钱，否

则就有可能会血本无归。

风险收益平衡不是什么新概念。小时候玩追人游戏的时候，如果我一直不停地跑到中间，大家都会觉得吃惊，但这样做就没人能追到我。这一概念充斥在我们的日常生活中。你要去参加一个会议，可已经迟到了，在考虑如何选择一条最快的路线时，就用到了风险收益平衡。你会想，如果交通畅通，我从第 32 大街走的话，就可以提前 15 分钟到；如果交通拥堵，我就彻底没希望了。

在有目的地选择投资哪种技术和领域时，风险收益平衡是一个很重要的权衡因素。15 年前，学会如何用 COBOL 编程是一项低风险的投资。那个时候，COBOL 程序员的竞争很激烈，平均工资并不高。掌握这门技术，你很容易就可以找到工作，但这份工作的经济回报较低。这就是低风险，低回报。

同样在那个时期，如果你选择学习了 Sun 公司的新语言 Java，或许你不能轻易找到工作，因为那时候使用 Java 编程的公司很少。谁都不知道 Java 到底能用来做什么。

但是如果在那一时期你仔细观察这个行业，就像 Sun 公司一样，你或许会发现 Java 的特别之处。你可能会预感到 Java 一定会火。投资越早，你就越有可能成为这个新技术潮流的领导者。

这样，你的决定就是正确的。如果你做事用心，恰到好处，那你在 Java 上的投资会给你带来可观的收益，也就是我们所说的高风险，高回报。

还是 15 年前，假如你看到了 Be 公司新产品 BeOS 的演示，那个时候这是个令人赞叹的产品。利用多处理器技术，这项产品强大的多媒体处理能力令人震惊。这个平台一鸣惊人，评论员们也开始头晕目眩，预测这项技术必将成为操作系统中的有力竞争者。有了这个新的平台，新的编程方法、新的 API 和新的用户界面概念也就应运而生了。要学的东西很多，但是看起来这些努力似乎都是值得的。你倾注了大量的努力来成为第一个创造 FTP 客户端，或者是第一个创造 BeOS 个人信息管理系统的人。当 Be 公司刚发行了与 Intel 兼容的操作系统时，就开始有传言说 Apple 要收购这家公司，使用它的技术作为新一代 Macintosh 操作系统的基础。

但结果是 Apple 并没有收购 Be 公司。事实上，Be 公司的产品就连高度专门化的小市场也没能打进去。这个产品没有得到进一步发展。那些为 BeOS 环境编程的开发人员慢慢痛苦地认识到，从长远看，他们的投资不会得到回报。最后，Be 公司被 Palm 收购，这个操作系统也无疾而终。BeOS 是一项高风险但是极具吸引力的技术投资，但是对那些投资者来说，这项新技术并没有给他们带来具体的长远收益。这就是高风险，零收益。

现在，我已经谈论了选择一项全新但是不稳定的技术和选择稳定成熟的技术的不同之处。选择一项已经进入商业生产流程的稳定技术，投资风险很低，但是与投资那些无人开发

的很炫的新技术相比，收益也会比较低。那么，那些即将完成使命的技术呢？只需轻轻一推，这些技术就跌进了坟墓。

那谁又是推动者呢？你或许会想到最后仅剩的几位 RPG 程序员，他们都已头发花白，数着日子等着退休。而新一代的程序员可能听都没听说过 RPG，他们学的都是 Java 和.Net。不难想象，一项陈旧的即将被淘汰的技术，它仅存的几名拥趸的职业生涯走向结束的过程，和这项技术本身走向终结的道路是一样的。

但是，旧的系统不是灭亡，而是被取代。在新旧交替的过程中，旧的系统需要与新系统对话。必须有人知道如何将新系统与旧系统融合，反之亦然。但是一般来说，新一代的程序员和那些即将退休的老程序员都不知道或者很想知道如何才能将两代系统的特点很好地融合起来。

所以，这就需要精明的技术人员来充当"技术收容所"的角色--帮助旧系统舒服且有尊严地消失。这项工作的重要性是绝对不能被低估的。就像大多数人在沉船之前会跳海一样，那些老的程序员要么就干脆退休，要么就向另一技术领域跨一步。作为一项仍然重要的技术的最后支持者，你当然是权威。但这也是极具风险的，一旦这个技术彻底退出游戏，那你就成了一种根本不存在的技术的专家了。但是，如果你行动得够快，还可以选择下一个正在衰退的系统，然后再来一遍。

选择是把双刃剑，决定权还是在你手里。

练　习

基于当今市场，按照从左往右的顺序尽可能多地列举出处于早期、中期和晚期的技术。最左边为崭新的尚未稳定的技术，最右边为即将退出市场的技术。尽可能仔细地找到它们之间的细微关联。

当你列举出所有你能想到的技术后，标记出你认为自己擅长的技术，然后换一种颜色，标记出那些你做过但是并不精通的技术。你的标记主要集中在哪个区域？它们是聚集，还是分散的？处于这张图表边缘处的技术，有没有你感兴趣的？

第 1 章 选择市场

你马上就要进行一次大的投资，也许并不是要投入大笔金钱，而是时间，是你的一生。大都数人对待工作的态度往往都是顺其自然，走一步看一步--我们刚刚深入了解了 Java 或者 VB，老板有一天突然参加了一个热门技术的培训，于是我们就转而学习新技术，直到有人又把新的东西递到我们手里。我们的职业道路就是由一连串没有方向的偶然构成的。

在《程序员修炼之道》一书中，Dave Thomas 和 Andy Hunt 谈到了编程中的偶然性。下面这个场景，会引起大部分编程员的共鸣：当你开始做一个程序的时候，或许手头上有一个

从网上复制的示例程序，看上去这个程序可以使用。为了满足你的需要，你会对这个程序稍加改动--添加一些代码，再加一点。你根本就不知道自己在做什么，只是不断地做一些小的修改，直到这个程序完全满足你的需要。但问题是，这样做就像是用纸牌搭建房子，每增添一张纸牌，就增加了一分纸房子坍塌的危险。你根本就不知道这个程序是如何工作的，所以你每做一点儿改动，都有可能导致你的程序完全失败。

作为软件开发人员，用这种投机取巧的方式来编程显然不是什么好主意。但是很多人正是让偶然来决定职业道路上的各种选择。我们应该在哪种技术上投资？应该专注于哪个领域？是应该扩展知识面，还是深入学习一门学问？这些问题都是值得我们细细斟酌的。

想象一下你开了一家公司，现在正要生产你们的明星产品。如果这个产品失败了，公司就会破产。你会花多少精力来思考此产品的消费者是谁？在产品进入生产流程之前，你又会用多少时间来弄明白这个产品到底是什么？我相信你肯定会仔仔细细地考虑其中的每个小细节，然后亲自做出决定。

但是，在职业道路上，面临选择的时候，我们为什么就缺少了这番心思呢？如果你把自己的职业当成是一门生意（事实上它就是一门生意），那么你的"产品"就是由你提供的服务构成的。这些服务是什么？你又会把它们出售给谁？接下来的一年，对此种商品的需求是会增加还是减少呢？在这些选择上你愿意投下多少赌注？

读完本章的内容后，你会找到答案。

1　稳定成熟的技术还是未成熟的新技术？

如果你想投资，可以有许多方法。你可以把钱存进银行，但是利息的增长往往跟不上通货膨胀的速度。买国债也是个办法，但同样，收益也不会很高。不过，这两种投资方法都无需承担什么风险。

你也可以选择把钱投入一个小规模的创业公司。投入几千美金换取公司的一小部分股份。如果公司的决策正确，而且这个决策被有效地执行了，那么你就有可能挣一大笔钱，否则就有可能会血本无归。

风险收益平衡不是什么新概念。小时候玩追人游戏的时候，如果我一直不停地跑到中间，大家都会觉得吃惊，但这样做就没人能追到我。这一概念充斥在我们的日常生活中。你要去参加一个会议，可已经迟到了，在考虑如何选择一条最快的路线时，就用到了风险收益平衡。你会想，如果交通畅通，我从第 32 大街走的话，就可以提前 15 分钟到；如果交通拥堵，我就彻底没希望了。

在有目的地选择投资哪种技术和领域时，风险收益平衡是一个很重要的权衡因素。15 年前，学会如何用 COBOL 编程是一项低风险的投资。那个时候，COBOL 程序员的竞争很激烈，平均工资并不高。掌握这门技术，你很容易就可以找到工作，但这份工作的经济回报

较低。这就是低风险，低回报。

同样在那个时期，如果你选择学习了 Sun 公司的新语言 Java，或许你不能轻易找到工作，因为那时候使用 Java 编程的公司很少。谁都不知道 Java 到底能用来做什么。

但是如果在那一时期你仔细观察这个行业，就像 Sun 公司一样，你或许会发现 Java 的特别之处。你可能会预感到 Java 一定会火。投资越早，你就越有可能成为这个新技术潮流的领导者。

这样，你的决定就是正确的。如果你做事用心，恰到好处，那你在 Java 上的投资会给你带来可观的收益，也就是我们所说的高风险，高回报。

还是 15 年前，假如你看到了 Be 公司新产品 BeOS 的演示，那个时候这是个令人赞叹的产品。利用多处理器技术，这项产品强大的多媒体处理能力令人震惊。这个平台一鸣惊人，评论员们也开始头晕目眩，预测这项技术必将成为操作系统中的有力竞争者。有了这个新的平台，新的编程方法、新的 API 和新的用户界面概念也就应运而生了。要学的东西很多，但是看起来这些努力似乎都是值得的。你倾注了大量的努力来成为第一个创造 FTP 客户端，或者是第一个创造 BeOS 个人信息管理系统的人。当 Be 公司刚发行了与 Intel 兼容的操作系统时，就开始有传言说 Apple 要收购这家公司，使用它的技术作为新一代 Macintosh 操作系统的基础。

但结果是 Apple 并没有收购 Be 公司。事实上，Be 公司的产品就连高度专门化的小市场也没能打进去。这个产品没有得到进一步发展。那些为 BeOS 环境编程的开发人员慢慢痛苦地认识到，从长远看，他们的投资不会得到回报。最后，Be 公司被 Palm 收购，这个操作系统也无疾而终。BeOS 是一项高风险但是极具吸引力的技术投资，但是对那些投资者来说，这项新技术并没有给他们带来具体的长远收益。这就是高风险，零收益。

现在，我已经谈论了选择一项全新但是不稳定的技术和选择稳定成熟的技术的不同之处。选择一项已经进入商业生产流程的稳定技术，投资风险很低，但是与投资那些无人开发的很炫的新技术相比，收益也会比较低。那么，那些即将完成使命的技术呢？只需轻轻一推，这些技术就跌进了坟墓。

那谁又是推动者呢？你或许会想到最后仅剩的几位 RPG 程序员，他们都已头发花白，数着日子等着退休。而新一代的程序员可能听都没听说过 RPG，他们学的都是 Java 和.Net。不难想象，一项陈旧的即将被淘汰的技术，它仅存的几名拥趸的职业生涯走向结束的过程，和这项技术本身走向终结的道路是一样的。

但是，旧的系统不是灭亡，而是被取代。在新旧交替的过程中，旧的系统需要与新系统对话。必须有人知道如何将新系统与旧系统融合，反之亦然。但是一般来说，新一代的程序员和那些即将退休的老程序员都不知道或者很想知道如何才能将两代系统的特点很好地融

合起来。

所以，这就需要精明的技术人员来充当"技术收容所"的角色--帮助旧系统舒服且有尊严地消失。这项工作的重要性是绝对不能被低估的。就像大多数人在沉船之前会跳海一样，那些老的程序员要么就干脆退休，要么就向另一技术领域跨一步。作为一项仍然重要的技术的最后支持者，你当然是权威。但这也是极具风险的，一旦这个技术彻底退出游戏，那你就成了一种根本不存在的技术的专家了。但是，如果你行动得够快，还可以选择下一个正在衰退的系统，然后再来一遍。

选择是把双刃剑，决定权还是在你手里。

练　习

基于当今市场，按照从左往右的顺序尽可能多地列举出处于早期、中期和晚期的技术。最左边为崭新的尚未稳定的技术，最右边为即将退出市场的技术。尽可能仔细地找到它们之间的细微关联。

当你列举出所有你能想到的技术后，标记出你认为自己擅长的技术，然后换一种颜色，标记出那些你做过但是并不精通的技术。你的标记主要集中在哪个区域？它们是聚集，还是分散的？处于这张图表边缘处的技术，有没有你感兴趣的？

2　供应和需求

Web 被广泛使用后，你只需为公司创造一个简单的 HTML 就能挣不少钱。每个公司都想拥有自己的网站，但很少人知道怎么制作。各家公司都愿意高薪聘请有经验的网页设计师。那时候只需知道基本的 HTML、超链接和站点结构，就可以称为有经验的 Web 设计师了。

制作 HTML 非常简单。制作出好的网页不容易，但是基础的东西很好掌握。那时候，Web 设计师供不应求，工资极具诱惑力，越来越多的人开始阅读相关书籍自学 HTML。结果，越来越多的人成为 HTML 方面的专家。

当 Web 设计师越来越多时，就开始划分真正具有艺术性的设计师和实用主义设计师。竞争也降低了他们的薪酬。由于雇佣 Web 设计师价格低廉，越来越多的公司开始要创建自己的网站。以前他们或许要付 5000 美金才能制作他们的第一个网站，现在只需付 500 美金。

当然，也有公司仍然愿意花大价钱制作出色的网站。那些优秀的设计师也有资本开出高价钱。

最终，网页设计师的薪酬降到了中低水平。一般水平的 Web 设计师逐渐被最终用户以及做 IT 但并非专业做网页的人取代。这样，HTML 设计者的供应、需求和价格达到了平衡。

Web 设计师行业的历史发展证明了一个众所周知的经济规律--供求规律。提到供求，大都数人都会想到一件商品的价值是多少，应该卖多少钱。如果市场上这种商品供大于求，价

格就会下降；如果供小于求，那么价格就会上涨。

除了可以预测商品和服务的价格，供求关系的规律还可以预测价格的变化将如何影响出售和购买此种商品或服务的人数。通常，同一件商品的价格越低，购买者越多。

这条规律有什么价值呢？我们可以把编程工作外包给国外团队，将大量的廉价 IT 工作人员注入到我们的市场经济中。在国内，我们担心失去工作，但是廉价的劳动力事实上也增加了市场对 IT 人员的总体需求。同时，随着需求的增加，价格也在降低。高需求产品和服务的竞争是以价格为导向的。在买方市场，价格就是薪水。你不能在价格上与他们竞争，因为你承受不起，那怎么办呢？

国外市场为我们的市场注入了廉价的开发人员，但是涉及的技术范围很窄。印度有很多的 Java 和.NET 程序员，也有很多 Oracle DBA。在国外从事非主流技术的人员还是很少的。当选择专注于哪种技术的时候，你要仔细考虑供给增长和价格下降给你的职业前景带来的影响。

作为.NET 程序员，你会发现自己每天都在和成千上万的人竞争。但如果你是 Python 程序员，那么竞争就小得多。这会造成.NET 程序员的平均工资大幅降低，也就可能会引起市场需求的增长，也就是说，会产生更多的.NET 工作机会。这样，你可以很快地找到一份工作，但是薪水不会令人满意。相对于市场需求来说，Python 程序员的供给比.NET 少得多。

如果 Python 工作能提供更高的薪水，那么就会有更多的人为了追求更高的薪水来做这份工作，这样就加剧了竞争，也会降低 Python 程序员的薪水。

这就是供求平衡。但到目前为止，印度专门为已经平衡的 IT 市场提供服务。在印度，主流的外包公司不会着手做新技术。他们从来都不做第一个吃螃蟹的人。他们等待技术服务市场平衡，然后再用极其廉价的编程成本打入这个市场。

这样说来，你可能会选择市场上需求较低的工作。如果你害怕失去工作，自然而然地，你就会选择避免与外包公司做相同的工作。既然外包公司的工作都是市场上需求较高的，那么你就应该关注那些特殊领域的技术。这样或许不能减轻竞争压力，但是竞争的重点会由价格转向能力--这正是你需要的。你无法在价格上与他们竞争，但是可以在能力上与之抗衡。

同样地，随着主流程序员平均成本的降低，需求就会增加。对 Java 程序员整体需求的增加，会导致国内工作机会的增加。国外廉价 Java 程序员的增加可以拉动市场需求，包括对更高级程序员的需求。

现实正是如此。许多公司看到要使国外团队更好地工作，它们就必须要留住国内那些更高级的程序员。这些高级程序员可以制定标准、保证质量、领导技术团队。市场对 Java 程序员整体需求的增长，会导致对此类高级开发人员需求的增长。低端工作可能会流向国外，但比起外包之前，市场上会多出更多的高端工作机会。与特殊技术市场的情况类似，从事高

端层面的 Java 开发工作，竞争就会从价格转到能力上。

从供求规律中，我们可以学到重要的一点--需求的增长会加剧价格的竞争。如果只想做稳定可靠的工作，并且跟随着工作发展，那么你就会卷入与国外开发人员的价格竞争中，因为你的技术决定了你只能进入平衡的外包市场。如果在主流技术市场中竞争，你就必须在更高层面上竞争，否则，你就要去发现市场上的不平衡，找到外包公司无能为力的工作。这两种情况，你都必须找到工作的动力，提高自身的技术和灵敏度来应对一切变化。

练　习

研究当今技术市场的需求。利用招聘广告和招聘网站找出哪些工作是高需求，哪些是低需求的。登陆外包公司的网站（如果你在这些公司工作，可以直接与员工交流），把这些公司的技术与你发现的高需求工作进行比较。记录下那些在国内市场中高需求且没有流到外包市场的技术。然后再将这些外包公司的技术与前沿科技相比较。密切关注外包公司还没有涉足的上述两类技术。思考它们需要多长时间才能为相应的市场提供服务。这个时间差就是市场不平衡的阶段。

3　只会编程是不够的

3　只会编程是不够的

只思考在哪种技术上投资是不够的。毕竟，技术只是一种商品。你不可能只掌握一种编程语言，或者只能够操作某种系统，然后把生意交给老板打理。如果他们只想找个懂代码的机器人，那不如雇个外国廉价的程序员。如果你想站稳脚跟，必须要深入了解你所处的领域。

事实上，软件工程师不能只会开发软件，应该要成为这个业务领域的专家。在我之前工作过的一家公司里，就有这么一个团队。我第一次见到这个公司数据库管理团队的时候有点儿震惊，因为这个团队里的成员都相当厌烦数据库技术。我当时在想，既然是这样，那这些人为什么要干 IT 呢？单在技术上讲，他们算不上出色，但是这个团队有他们的特别之处。作为企业数据的保存和维护人员，他们比那些商业分析师更加了解这个行业。他们的知识和对这个行业的了解使他们成为了数据管理工作的抢手人才。我们这些愚人居然还看不起人家。他们做的工作正是他们的价值所在。

你的行业经历应该成为你的重要才能。如果你是搞音乐的，当你描述你的才能时，不能只说我能演奏某首曲子，而要说你真正了解这首曲子的内涵。商业领域的经验也是一样。比方说，如果你正在做一个医疗保健项目，你能区分出 HIPAA835 和 HIPAA837 这两种电子数据交换（EDI）协议有什么不同吗？同是软件开发人员，这个知识不就能决定谁更适合这个职位了吗。

或许你只是一个程序员，但是如果你能用客户所处行业的专业语言与他们交流，那这就

是一项非常重要的技能。就像如果与你工作的人都真正了解软件开发是怎么回事，你会不会觉得一切都会变得更加得心应手呢？你再也不用向他们解释为什么不能在 Web 应用程序中的一个页面上返回 30 000 条记录，或者解释为什么他们不能向开发服务器发送链接。你的客户也有同感。换位思考一下，如果你是客户，为你服务的程序员了解你的行业，不用什么都得由你来做决定，你也不用紧张担心哪个小细节会出问题，你会不会觉得工作起来更容易呢？你从事的行业也是这样。

现在的软件开发界，Java 和.NET 大行其道。如果你会这两门技术，那你就能在使用这两项技术的公司找到工作。选择商业领域也是同样的道理。在选择从事哪个行业的时候，你应该像选择掌握哪门技术时一样谨慎。

鉴于行业选择是十分重要的，那么选择在哪个公司、哪个领域工作对你来说也是重要的。如果你还没有仔细考虑过这个问题，那现在开始思考吧。机遇每天都在流逝。就像利息马上就涨了，但你却把钱存在了一个低利率的死期账户里。把自身的发展限制在一个静止不前的行业里，可不是什么好的投资选择。

练　习

(1) 安排一次与业内人士的午餐，问问他们是如何工作的。交流中，思考如果你来做他们的工作，你会做什么改变或者你可以从他们身上学到什么。询问他们日常工作中的细节。问问他们技术是如何帮助（或者阻碍）他们工作的。从他们的角度出发，思考你的工作。

定期安排此类活动。刚开始你可能会觉得有些尴尬，但没关系。我是几年前开始这么做的，这极大地帮助我理解和融入我所服务的行业。另外，在与我的客户交谈时，我也变得更加得心应手。

(2) 选择一本与你公司行业有关的杂志。你甚至都不用买，大多数公司都有些过期的行业杂志。试着阅读它们，虽然有些东西你可能不懂，但是要坚持。列出你可以向客户询问的问题。不要担心你的问题很傻，客户会大为赞赏你的这种学习态度。

找一个你可以随时登录的行业网站。无论是浏览网站时，还是阅读杂志时，注意大事件和专题文章。你所处的行业正在为什么而努力？现在的热门是什么？不管是什么，把它们介绍给你的客户。请他们说说观点看法。思考这些潮流是如何影响你的公司、你的部门、你的团队，以及你自己的工作的。

4　做团队中最差的

4　做团队中最差的

爵士乐的传奇人物，爵士乐吉他手 Pat Metheny 给年轻音乐演奏者提出了一条建议--"做乐队中最差的乐手。"

进入 IT 这行之前，我是一名专业爵士和蓝调布鲁斯萨克斯风演奏者。作为一名乐器演奏者，我很幸运地早早就学到了这个道理并且一直坚持这么做。做乐队中最差的乐手意味着你总是在与比你优秀的人一起演奏。

这样的话，你为什么不选择做这个最差的乐手呢？你会问"那这样不就会承受很大的压力么？"没错，刚开始压力是很大。作为一名年轻的乐手，我总是十分显眼，因为我总是乐队中最差的乐手。去演出时我连萨克斯风都不想拿出来，因为我怕被人赶下舞台。那个时候我总是仰视身边的人，期望有一天自己也能达到他们的水平，甚至梦想能成为乐队主奏。

感谢上帝，我没有失败。神奇的事情发生了。我在这行占有了一席之地。我不是乐队中最差的那个，但也没有成为最优秀的。这有两个原因，其中一个原因是我并不是自己想象得那么差。这点我们稍后再讨论。

更有意思的原因是我的演奏可以自动模仿我的偶像演奏出来的音乐，这使我在他们中间占有了一席之地。我希望这是因为我自身具有某种超能力--站在一个天才旁边，就能拥有他的能力。但回想起来也没这么神奇，这好像就是出于一种本能。就好像如果我周围的人说话方式与我不一样，那我就会自然而然地受他们影响，说话时使用他们的词汇或者语法习惯。我曾在印度生活过一年半，从印度回来后，我妻子经常被我说的话逗得哈哈大笑，她问我："你听见自己刚才说什么了么？"我居然在讲印度英语。

我做萨克斯风手时，就是做乐队中最差的演奏者。我只能像其他人一样演奏。事实上当我在赌场或者巴掌大的酒吧里与那些差劲的乐队一起演奏时，我的演奏水平也向他们靠拢。我发现就算不是在酒吧演奏，我也摆脱不了从那些差劲的乐队那里染来的坏习惯。就好像那些酒鬼，清醒的时候说话也含糊不清。

所以我认识到人们会取得很大的进步或者退步，仅仅是因为与他们合作的人不同了。与一个团队合作的时间长了，会对自身的能力产生持久的影响。

作乐手的时候，我养成了寻找最好的乐手与之一起演奏的习惯。进入 IT 这行后，这种习惯自然而然地延续了下来。我下意识地去寻找最棒的 IT 人士，并与他们一起工作。显然，真理是禁得起考验的。做编程团队里最差的程序员和做乐队里最差的乐手产生的效果是一样的。你会发现自己变得出奇地睿智。你写的东西，和你的谈吐都会变得越来越有智慧。你编写的程序和设计会越来越高雅优美。你会越来越有创造力，难题也迎刃而解。

好，现在我们回到能让我意想不到地融入乐队的第一个原因。我确实不像自己想象的那么差劲。在音乐这行，要想得到别的乐手对你的真实评价，并不是件难事。你优秀，那人家就会再次邀请你合作；你差劲，别人就会避免和你合作。比起你直接问起他们如何评价你，这种检验方法更能得到真实的反馈，因为好的乐手不愿意和差劲的乐手同台。让我吃惊的是，很多优秀的乐手都会再次邀请我和他们同台，甚至邀请我与他们一起组建乐队。

试图做一个团队里最差的人可以让你不再小看自己。可能你的能力应该是在甲等乐团演出，但你自己却认为自己属于乙等乐团，这都是因为你恐惧。清楚地知道自己不是最好的，就不会总担心被人发现你不是那么优秀。事实上，即使你在尝试做那个最差的，也并不意味着你就是最差的。

练　习

找一个团队，让自己成为"最差"的。不需要立刻调换工作，你可以试着找一个志愿者项目，通过与这个项目中其他程序员的合作，提高自身能力。查查有哪些编程团队会议，然后去参加这些会议。程序员一般都会用业余时间做兼职，以此来练习新的技术，提高自身技能。

如果在身边找不到这样的程序员组织，就利用网络。找一个你钦佩的开源项目，且他的设计者是你下一阶段发展的目标。浏览这个项目的待处理列表和官方讨论区，或者编写一个功能或者修正一个大的错误。你的代码要模仿这个项目的代码风格，但是又要让你的代码和设计与其他项目完全不同，甚至让原作的程序员都认不出来。在你觉得一切都妥当之后，把它作为一个补丁提交。如果你做得好，这个项目就会接受它。这样重复来做。如果这个项目的设计团队不同意你的观点，那就将他们的反馈加入到你的设计中再次提交，或者记录下他们做出的改变。最终，你会发现自己成为了这个项目团队中值得信赖的一员。你会惊喜地发现虽然这些高级程序员并不在你的身边，你甚至连他们的声音都没听过，但你已经从他们身上学到了很多东西。

5　在思维上投资

5　在思维上投资

当你选择专注于哪个领域发展的时候，那些容易找到工作的技术很吸引眼球。Java 和.NET 都是很强大的。学习 Java，你就可以去申请一份编写 Java 代码的工作，而且成功得到这份工作的几率很高。

这样想，那如果在一种新的还未稳定的技术上花费时间和精力，就会显得很愚蠢，特别是如果你之前并没打算开发这种技术。

TIOBE Software 利用网络搜索引擎，根据全球范围内有经验的工程师、课程和第三方供应商对程序设计语言的实际使用率，将编程语言做出排序 。这种统计方式虽然不是很科学，但可以起到很好的指示作用。

当我还在撰写此书的时候，最受欢迎的编程语言是 Java，C 语言紧随其后，C#位列第 6 位，但是已出现微小的上升趋势。SAP 的 ABAP 位列第 7 位，但名次成缓慢下降趋势。我最喜欢的 Ruby 排名第 11 位。在重要的工作中我一般都用 Ruby，并用它来做每年国际性会议的议题。但当本书第一版发行时，Ruby 居然跌出了前 20 名，位于 ABAP 之后！

这么说来，我使用 Ruby 只能说明我疯了或者傻了？Paul Graham 在 Great Hackers 一文中曾宣称使用 Java 的程序员没有使用 Python 的程序员聪明，这一观点在这个行业中引起了一阵骚动。他惹怒了很多愚蠢（不敢相信我自己居然这么说）的 Java 程序员，他们在自己的网站上驳斥这一观点。这种反击行为恰恰证明 Paul Graham 触动了这个行业的一个敏感点。当他第一次以演讲的方式发表这篇文章时，我在现场，他让我回想起了往事。

一次我去印度招聘，要从上百名面试者里挑选十几名适合的人选。整个招聘团队筋疲力尽，因为大家费尽周折，却根本没有挑选到适合的人。我们头疼欲裂，眼睛也熬得通红，晚上开会商讨应该如何改变面试的策略。为了面试更多、更优秀的候选者，我们需要优化面试流程。我连续 12 个小时努力让那些紧张沉闷的应聘者开口说话，嗓子都哑了。所以我提议在猎头简历搜索库的关键词中增添"Smalltalk"，但是人力资源总监的答案是"在印度，没人知道什么是 Smalltalk。"这就是关键所在了。没人知道 Smalltalk，用 Smalltalk 编程与用 Java 是完全不同的。这种不同的经验使我们对候选人的期待值不一样， Smalltalk 环境的动态特征赋予 Java 程序员在处理问题时一种新的思维的方式。我希望这些因素能够让我发现技术成熟的程序员，但在这之前我还没找到符合条件的应聘者。

在搜索关键词里增加了"Smalltalk"后，大大缩小了候选范围。符合条件的应聘者真正理解什么是面向对象的程序设计。他们认识到 Java 不是能解决任何问题的万应灵药。他们中的大多数人真正热爱编程！招聘团队就像发现了未经打磨的钻石，心里想，前两个星期你们都干嘛去了！

由于他们很优秀，所以有资格提出条件。可惜，我们给出的薪酬有限，不足以吸引他们。大都数人都选择留在原来的公司或者继续寻找工作。尽管没能留住他们，但我们学到了宝贵的招聘经验：比起那些经验单一的候选人，我们更倾向于那些具有丰富经验的候选人。我认为优秀的程序员之所以寻找变化和多样性的工作，是因为他们喜欢学习新东西，或者是因为他们很清楚要想成为更加成熟、更加全面的程序员，就必须去学习新的技术、在新的环境下工作，获取新的经验。我认为这两方面因素都奏效。现在我仍然使用这个技巧来招聘程序员。

所以你与其千方百计地想要进入我的候选人范围，不如把精力放在学习以前没有使用过的技术上。

作为招聘经理，我认为判断你适合不适合一个职位的首要因素就是你是否对这行感兴趣。如果我知道你为了自身发展，或者更理想的是，你单纯因为兴趣而学习新的东西，我就会知道你热爱你的职业，把你的职业视为动力。当我问候选人有没有用过某种非主流的技术时，最不愿意听到的答案就是"没有人给我机会使用"。没有机会？从来也没有人主动给我提供过这种机会啊！机会是要自己争取的。

除了可以激励你，使你更加热爱工作，更重要的是，接触这些边缘技术和方法能让你更有深度、更加优秀、更具智慧，以及更具创造力。

如果你认为这样还不足以成为你学习新技术的原因，那或许你选错了职业。

练　习

学习一种新的编程语言。但不是从 Java 到 C#或者是从 C 到 C++。这门新的语言应该可以让你的思维方式产生变化。如果你是 Java 或者是 C#的程序员，那就尝试学习类似 Smalltalk 或者 Ruby 这种不需要采用强类型的静态编程方式的语言。或者，如果你一直在做面向对象开发的话，可以尝试 Haskell 或者 Scheme 这样的函数式语言。你不需要成为专家，可以感到这种新的编程环境与你之前所处的环境的不同之处即可。如果你觉得并没有什么不同，那就说明你选错了语言或者你仍然将固有的思维方式运用到新的语言中。要彻底改变你的思维方式来学习新的语言。向熟悉这些语言的程序员请教，让他们检查你的代码并提出建议，使之更符合此种语言的特性。

6　不要听从父母

6　不要听从父母

我们的文化要求我们听从父母的建议。这就是一个孩子的职责--做应该做的事情--就像是一种虔诚的信仰。图书、电影和电视情节都在讲述或上演着父辈的智慧。但在我们这个行业，这条真理是行不通的。

父母总不希望儿女去冒险，所以他们并不期望儿女有一个多么卓越的职业，只要差不多就行了。比起其他人的建议，父母给的建议总是包含着种种担心。这种出于担心的建议目的就是不要让你经历失败。但想着如何避免失败绝对不是取得成功的方法！成功是要冒险的。胜利者想的是他们想要做什么，而不是其他人会怎么做。出于担心的职业规划不会让你走向成功，而是会局限你的发展。没错，这条路很安全，但毫无乐趣而言。

在上一代人选择职业的时候，乐趣绝对不是一个决定因素。工作不是用来产生乐趣的，而是为了填饱肚子。工作之余才会谈到乐趣，那是晚上下班后和周末的事情。但是后来我们认识到，如果工作没有乐趣，那我们就没有动力去做好它。现在，这种观念虽然没有什么大的改变，但是我们的文化在如何看待工作的意义这个问题上向好的方向转变了。越来越多的人懂得了只有对工作充满激情，才会做出卓越的工作。在软件这行，如果没有乐趣，那工作起来就不可能充满激情。

另外一个决定职业规划的因素是跳槽，父母也不会赞成这个观点。一个成熟的职业软件开发人员需要从各个角度了解这个行业：产品开发、IT 支持、内部业务系统开发以及管理工作。作为软件开发人员，你看到的角度越多，攻克的技术难题越多，就意味着你越有足够的能力来面对艰难项目。对一个程序员来说，只在一个公司工作，加强单一业务技能，会局限职业发展。"铁饭碗"的时代已经不复存在了。以前，这种进入某个公司并为其服务终身的行为被看作是一种奉献。但现在这是一种障碍。如果你只在一个公司工作过，只看到了一种

系统，那么当那些明智的经理决定是否要雇用你的时候，它就成为了一个不利因素。就我个人而言，比起那些只知道一种做事方法的人来说，我更愿意聘请在不同的环境中经历过成功与失败的人。

几年前，我开始认识到我父母那一代人的职业价值观很大地影响了我的职业规划。那时候我受雇于全球规模最大且稳定的公司之一，并且慢慢稳步地升迁。但我发现我好像没有什么前途。我安慰自己说这个公司这么大，我的发展不会受到局限。我可以在不同的工作地点做不同的工作，但是最终，我还是在同一个地方做着相同的工作。

我曾经和一个朋友提起过想换个公司，他却说："没准你下半辈子注定要在大公司工作。"上帝，不要啊！于是我马上找到了一份新的工作，离开了原来的公司。

这也是我在软件界走向成功的标志性起点。我看到了以前从来没有见过的领域，我开始应对更难的问题，我得到的回报比以往任何时候都多。刚开始我确实有点害怕，但是当我改变了出于担心的保守职业规划后，我的事业和我的生活都变得更加美好了。

在职业道路上，需要一些有目的性的冒险。别让恐惧征服了你。如果在工作中没有感到乐趣，那你就不可能出色地工作。

练　习

在职业道路上，你最担心什么？回想你最近做过的几次职业选择，不用是很大的决定（如果你是出于某种担心而做出职业的选择，那也不会是什么大决定）。你从事了什么特殊的工作，或者你申请了一份新的工作或升职。把这些选择罗列出来，逐一做出诚实的评价：这些决定受到"担心"这一因素的影响有多少？如果你没有担心，那你会做到什么程度？如果这些决定确实是受到"担心"的影响了，那你现在如何逆转它，寻找新的机会做出新的选择，当然这次不要再因为担心什么而受到约束。

在微软 30 万美金的诱惑前，我却选择了 GitHub

--Tom Preston-Werner，GitHub 的创始人之一

2008 年是闰年，也就是说在 366 天前，几乎不差一分钟，我一个人坐在旧金山第三大街的 Zeke's 运动酒吧的包间里。我不经常去运动酒吧，但那个周四是"我可以谈论 Ruby"之夜。我当时想"我可以……"总是个好事。ICHR 是个半私人的会议，来的人都是志同道合的 Ruby 爱好者，这个聚会一般都会延续到晚上，大家通宵达旦地饮酒。通常这种晚会会如同我第二天早上的宿醉一样烟消云散，但是这个夜晚不同，因为它是 GitHub 诞生之夜。

一开始我说自己一个人坐在包间里是因为那天我们的长桌在酒吧靠后的地方，灯光昏暗，我刚点了杯 Fat Tire，需要跳出那个社交氛围休息一下，所以感觉就像是一个人坐在包间里。我小饮了几口，Chris Wanstrath 走了进来。我现在想不起来那时候我和 Chris 是不是

已经成为了朋友。我和他在关于 Ruby 的聚会和会议上见过面，但只是泛泛之交。我也不记得当时为什么就朝他做了个手势让他过来，可能是因为我觉得他编的程序很不错。我对他说："老兄，看看这个。"大约一星期前，我刚开始做一个叫做 Grit 的项目。这个项目使我可以通过 Ruby 代码以一种面向对象的方式访问 Git 的代码库。Chris 是当时很少的几个开始认真关注 Git 的 Ruby 专家。他坐下来，我开始把我所做的工作展示给他。我做的东西不多，但显然，它已经足以激起 Chris 的兴趣了。看出这点后，我开始向他讲述一个不成熟的设想：我想做一个类似网站的东西，这个网站作为中心，程序员可以在上面分享他们的 Git 代码库。我还给它起了个名字--GitHub。当时可能我还在解释什么 Chris 就打断我说："算我一份，就这么干！"而且说得异常坚定。

第二天--2007 年 10 月 19 日星期五，晚上 10 点 24 分--Chris 带来了他的第一笔投资，用数字石雕为我们的合伙事业盖上了印章。到那时为止，我们两个人还只是想在一起编程，没讨论过这个计划要如何进行。

还记得电影《小子难缠》（The Karate kid）中，丹尼被训练成武术大师的情节么？那几分钟非常精彩。记得那段音乐么？或者你应该去听听 Joe Esposito 的 You're the best，因为接下来的故事非常蒙太奇。

接下来的三个月里，我和 Chris 花了大把的时间在 GitHub 的设计和编程上。我继续研究 Grit，设计出了用户界面。Chris 构建了 Rails 应用。我们每周六见面讨论设计方案，试图给这个项目做出预算。一个大雨天，我们俩吃着美味的越南蛋卷，就定价策略这个问题谈论了两个多小时。那时候，我们俩还都有各自的正式工作。我在 Powerset 公司为 Ranking 和 Relevance 团队开发工具。

3 个月没日没夜的工作之后，到 2008 年 1 月中旬，我们推出了一个私人试用模式，并向我们的朋友发出了邀请信。2 月中旬，P.J. Hyett 加入了我们，壮大了我们的团队。4 月 10 日我们正式启动了这个网站，但没有通知 TechCrunch 网站。这时候，这个项目还只是由三个 20 多岁的小伙子创立的事业，没有半毛钱的外界投资。

2008 年 7 月 1 日，我还在 Powerset 做着全职工作，那天微软公司花费大约 1 亿美金收购了 Powerset。这就有趣了。我之前就知道早晚得做出去留的抉择，但没想到会这么快。我要么与微软签合同成为他们的一名员工，要么就辞职，全职做 GitHub。那年我 29 岁，是 3 个人中最大的，也是 3 人中欠债份额最多、月消费最多的一个。我已经习惯了年薪 6 位数的生活方式了。另外，我妻子在哥斯达黎加的考察工作马上就要结束了。我很快就要从一个貌似单身汉的身份回到已婚男人的生活。更让我难以做出决定的是，微软开出的薪酬十分诱人--薪水再加上工作超过 3 年奖励 30 万美金。这么具有诱惑力的条件我相信任何人都会三思而后行。所以当时我面临的选择是：一份在微软稳定且高薪的工作，或者不知道要投入多少资金并且极具风险的事业。另外两位伙伴在有了些积蓄后，他们就不再做全职工作了，全身

心地投入于 GitHub。所以我知道我待在 Powerset 的时间越长，对另外两个伙伴来说压力就越大。这是决定"做或者放弃"的关键时刻。选择 GitHub 并为之奋斗，或者做个安全的选择--在微软工作，让荷包满满，放弃 GitHub.

如果你想要个睡不好觉的秘方，我这有：把"我妻子会怎么看待这个问题"加上 3000 张百元美钞中，再加上你最爱的啤酒，最后放入一个清偿债务的诱惑。

如今，对于如何向老板提出辞职，我已经得心应手了。那天当我接到继续受雇的通知时，我告诉老板我要辞职，全力去搞 GitHub。我的老板很好，他虽然有些吃惊但还是对我表示理解。他没有用更高的奖金来诱惑我。我觉得他心里知道我是非走不可了。因为我想要离开，所以比起留在这里，新机会给我的诱惑更大。微软的经理们是非常狡诈的。他们对如何给出保留奖金相当在行--因为我想要离开，所以比起其他人，新老板给我的诱惑可能更大。告诉你，微软的经理非常精明。他们对如何提供保留奖金非常在行。当身边有这么一个人的时候，事情就会变得有些古怪。

最后，就像印第安纳琼斯永远不会放弃寻找圣杯的机会一样，就算另一个选择再稳妥，对于我真正热爱的事业，我也绝不会放弃。等我老了，驾鹤西游之前，回想过去我希望我会说"上帝，这辈子真是险象环生啊！"而不是 "嗯，这辈子过得还算稳稳当当。"

7　做一名通才

7　做一名通才

至少 20 年前，绝望的经理和老板一直欺骗自己说软件开发是一种机械化的生产流程。制定出规格标准，架构师把这些规格标准转化为高层次的技术层面。设计师再把详细的设计文档填入这个框架，然后交给像机器人一样工作的程序员，他们一只手拿着低俗小说，另一只手慵懒地敲入设计执行方案。最后，Inspector 12 收到完整的编码，经测试符合最初的规格标准后，准许通过。

经理们都希望软件开发机械化，这并不是什么奇怪的事。他们了解怎么做好机械化工作。几十年的经验教会了我们如何有效精确地制造有形的东西。所以，把我们从机械生产中学到的经验运用到软件开发上，我们就可以将其优化成一系列的生产流程。

在这个所谓的软件工厂中，雇员都是专才。他们坐在流水线旁自己的座位上，把 Java 的部件组合在一起，或者在软件车床上打磨一个 VB 的应用程序。Inspector 12 是测试员。软件组件随着流水线向下流动，Inspector 12 每天以同样的方式测试这些组件并盖章以示合格。J2EE 设计师设计 J2EE 的应用，C++的编码师在 C++环境下编码。这个工厂中，一切都划分得很分明，安排得有条有序。

但是，这个类比并不成立。软件至少应该适应软件需求。这个行业已经变了，商业人士

知道软件很"温柔"，可以根据需求做出改变。但这也就意味着构建、设计、编码和测试环节也要相应地变得更加灵活，这些都是机械化生产过程无法满足的。

在变化如此迅速的环境下，灵活才能制胜。聪明的生意人在碰到难题时，会向身边的专业软件师寻求帮助。那么，你怎么才能成为这些生意人遇到困难时首先想到的"英雄"呢？答案就是--能够解决一切可能出现的难题。

但是这些难题是什么呢？没错，你我都一样，我们无法预测会产生什么难题。我只知道这些问题非常多样化，诸如严重的设计缺陷需要立刻修补，异构系统的集成，以及 ad hoc 报告的生成。面对这么多种问题，可怜的 Inspector12 可能会命不久矣了。

有句话是"什么都懂点，但什么都不专"，一般来说，这句话是贬义的，是说这个人没有专注于某一项领域，并深入学习，成为这方面的专家。但是，当你的购物网站"提交订单"出了故障，每个小时你都会损失上百个订单时，那这个"什么都懂点，但什么都不专"的人可能既知道这个程序代码是怎么运行的，还会做些简单的 UNIX 调试，分析 RDBMS 规范中潜在的性能瓶颈，并能检查网络路由器配置看是否存在某些隐蔽的问题，更重要的是，找出这些问题后，这个人可以很快做出架构和设计决定，纠正代码，部署一个新的系统。这样看来，机械化生产模式看起来就非常奇怪，而且具有很多的缺陷。

另外一个机械化生产模式无法立足的原因是：这种生产线使工作按照稳定的步伐直线进行，而软件项目通常是具有循环性的。不仅项目的流动是循环的，一个项目内部的工作也是循环的。在制定完软件的规格、架构和设计之前，程序员要么坐在椅子上等，要么在这段时间着手做其他项目。但这种同时参加多个项目的问题是，不管这个软件的开发目的是什么，当程序员要大展身手的时候，必须要依赖前后流程和经验。规格、架构和设计文档可能非常出色，但是如果程序员不懂这个系统是用来做什么的，他就不能很好地实现这个系统。

当然，我所说的不仅适用于程序员。软件开发上的任何一个职位都是如此。由于前后流程的问题，同时参加多个项目并不可行。结果是，我们的生产系统是低效率的。在机械化生产模式中，有各种各样的方法尝试解决效率低下的问题。但是，我们还没有想出办法来优化我们的软件工厂，使它变得更有效率。

如果你只是一名程序员、测试员、设计师或者架构师，那你很可能会坐在那里无所事事，或者当你的项目快要结束时，你却在忙忙碌碌。如果你只是一个 J2EE 程序员或者是一个.NET 程序员，或者是 UNIX 系统管理员，那当一个项目或者一个公司的关注点开始转移出你擅长的技术领域时，你就会发现你不再发挥作用了。这不是说在一个项目的流程中，你的价值有多大（架构师的价值往往最大），而是说你可以在多广的范围内发挥作用。

如果你想在这个行业站稳脚跟，那我建议你要成为通才。如果你害怕你的部门裁员，那你就该知道精简团队的时候，一个只会测试或者只会编码的人肯定会被裁掉的。如果你就是单纯地想要卓越，那更好，你要动动脑筋掌握大局。

成为通才就是说让你不要只专注于一种技术。在工作中，有很多方法可以让我们扮演多种角色。为了使成为通才这个概念形象化，我们可以把 IT 职业分解成几个独立的部分。我想到了五个，但肯定还有更多，就看你是如何划分了：

职业阶梯的各层

平台和操作系统

代码和数据

系统和应用

业务和 IT

这些不同的方面可以帮助你了解如何成为一名通才。这只是审视职业的其中一种分类方法，你可以针对自己的情况，找到更好的方法。这里我们只对这个分类进行讨论。

首先，你可以选择成为一名团队负责人、经理、技术人员，或者一名架构师、程序员、测试员。很多人都不明白能够适应和胜任不同角色的价值所在。例如，一名强大的团队领导者应该尽力成为多面手。现在国内的编程团队十分精简，团队领导应该既能领导团队做项目，又能在外包团队偷懒的时候，卷起袖子亲自修复紧急严重的漏洞。软件架构师也一样，他要是再能写一些代码，那可能会大幅度地提高整个项目的进程。当工作要跨越职业阶梯的等级时，人们不是不愿意去做，而是没有能力去做。程序员不会领导团队，团队领导人不懂编程。能够把两样做得都很好的人，太稀有了。

下一个要说的是平台和操作系统。现在如果一个做 UNIX 的人拒绝做 Windows，那就太不实际了。同样，做.NET 的也不可能不做 J2EE，任何基础平台都是这样。要想在这行站稳脚，就必须做个多面手。任何人都有自己喜欢的技术，但是我们不能太理想化，自己喜欢什么就做什么并不实际。现状是我们要成为某一项技术的专家，同时还应该再擅长几种别的技术。技术平台只是一种工具，你的技术必须要高于它。如果我们想雇一个只做 Windows 的人，那我们会去国外找。如果我们想找个真正了解 Windows 和 UNIX 开发，又能帮助我们把这两者结合起来的人，我们会在国内寻找。这就是团队精神的本质。

同样，软件开发师和数据库管理员（短短 10 年间，这个职业从无发展到现在的重要地位）之间的界限也不应该划分得那么清楚。数据库管理员应该既知道如何使用 GUI 管理工具，也知道如何创建一个特定数据库产品。你不需要非常了解如何使用数据库。另一方面，软件开发师越来越忽视了解如何使用数据库了。两者相辅相成。

我刚进入这行时，首先让我感到吃惊的是，很多受过良好教育的程序员居然不知道如何安装他们用来开发和部署的系统。与我合作过的一些开发人员居然不会在 PC 机上安装一个操作系统，更不会安装他们用来部署应用程序的应用服务器。一个真正懂得他工作平台的开

发人员简直太少见了。要能找到这么一个人才，那做出的应用程序就会更好，工作进展也会更快。

最后，我们在本章第 3 节中提到的存在于业务和 IT 之间的那堵墙，应该立刻被推倒。现在就开始学习你的行业是如何运作的吧。

练　习

列出你能将你的知识和能力融合在一起的工作内容。写下每个方面中你的专长。例如，如果你列出了平台和操作系统，那就可以在旁边写上 Windows 和.NET。在你专长的右边，再列出你要学习的一种或几种技术，可能是 Linux 和 Java（或者是 Ruby、Perl）。

然后尽快（一周之内）找出 30 分钟开始研究你要学习的一门技术。不要只是单纯阅读相关的书籍资料，动手实践一下。如果它是种网络技术，那就下载一个 Web 服务器安装包，然后自己安装。如果是与做生意有关的话题，那就找一个你的客户，约他出来吃饭聊聊天。

8　成为一名专家

8　成为一名专家

我问："仅仅使用 Java，如何编写一个程序使 Java 虚拟机崩溃？"应试者沉默了，然后无助地问："怎么可能？"

"抱歉，我没听清楚，您能再重复一遍问题吗？"这个声音听起来有些绝望。从我的经验来看，重复这个问题也起不了什么作用。但我还是提高声音，放慢语速，重复了一遍这个问题："仅仅使用 Java，如何编写一个程序使 Java 虚拟机崩溃？"

"……抱歉，我之前没这么做过。"

"我知道你没做过。那接下来这个问题呢--如何编写出一个程序，不使 Java 虚拟机崩溃？"

此次面试的目的是寻找真正优秀的 Java 程序员。面试一开始我就让这名应试者（包括我本周面试过的其他人）给自己打分，满分 10 分。他打出的分数是 9 分。我要找的是一名新星。如果一个人对自己的评价如此之高，那他干嘛不干脆编一个恶意程序，引发 Java 虚拟机崩溃？

应试者缺乏技术深度。

这就是一个声称是 Java 专家的人给出的答案。如果你在一个聚会上碰到他，问起他是做什么工作的，他会说："我是 Java 程序员。"但是，他却连这么简单的问题都答不上来，甚至连一个错误答案都给不出来。在这次紧张的全国招聘中，这种情况不是个例，而是非常普遍的。成千的 Java 程序员申请了职位，但没有一个人知道 Java 类装载器是如何工作的，

也没人能高度概况出 Java 虚拟机是如何处理内存管理的。

没错，你不需要知道这些知识，在别人的监视下编写基本的代码。但是这些人是所谓的"专业人士"。很多人都认为专于某种技术，就简单地意味着不知道其他的技术。要是这么说，那我就可以说我妈妈是一个 Windows 专家，因为她从来没使用过 Linux 或者 OS X。我还可以说我那些住在阿肯色州乡下的亲戚是乡村音乐的专家，因为他们从来都没听过别的类型的音乐。

假设你左臂下方的皮肤里长了一个奇怪的肿块，你去看家庭医生。你的医生建议你去看专科医生做个切片检查。如果这个专科医生在医学院学习的时候根本没上过课，或者他在做住院实习医生的时候没有做过你要做的这类切片检查呢？我的意思不是说他们可以从事比这个切片检查更高深的工作，要是他们只是学到了肤浅的知识，其他什么都不知道呢？你可能会问："手术过程中，监测仪器开始哔哔作响怎么办？"这个医生的答案是："以前没发生过这种情况，这回也不会发生的。我也不知道这仪器是干嘛用的，不过它从来没发出过哔哔声。"

谢天谢地，软件开发师做的不是性命攸关的工作。如果他们犯了错误，一般也就是导致项目超出预算，或者是造成产品缺陷使得他们的老板付出更多的金钱，而不是生命。

遗憾的是，软件开发界有很多这样肤浅的专业人士，这些人以"专业人士"为借口，只知道一门技术。在医学界，专科医生是指对某一特定领域有深刻了解的人。医生建议他们的病人向专科医生寻求帮助，因为在某些特定情况下，相比于全科医生，专科医生可以让病人得到更加专业的治疗。

那么，在软件界，什么样的人才能称得上是专业人士呢？我在招聘的时候找遍了每一个角落，寻找真正深刻了解 Java 编程和部署环境的人。我想要寻找的人是已经处理过我们工作中可能遇到的 80%的问题，并且拥有足够的知识来应付另外还未出现的 20%的问题。我需要的人是不仅可以处理高水平的抽象，同时应该了解那些实现高端抽象的低端细节。我需要那些可以解决部署问题的人，或者如果他们解决不了，至少应该知道找谁来帮忙的人。

计算机界变化迅速，只有这样的专业人士才能生存下来。你是.NET 专业人士，但这绝不能成为你除.NET 之外对一切一无所知的借口，而是说，你是.NET 的权威，但当 IIS 服务器需要重启时，对你来说是小菜一碟。有人问你怎么用 Visual Studio .NET 进行源控制集成时，你的答案是："我做给你看。"由于不满意应用性能，客户提出要退出项目，这时候，你只需要三十分钟就能把问题解决。

如果你做不到以上这些，那以后请不要再顶着专业人士这个头衔。

练　习

(1) 你是否使用在虚拟机上编译并执行的编程语言？如果你使用，花点时间学习虚拟机

内部是如何工作的。很多书籍和网站都专门就 Java，.NET 和 Smalltalk 进行讨论。学习这些东西总比你凭空想象要简单。

不管你使用的编程语言是不是依赖虚拟机，花点时间学习编写源文件。你敲打出来的代码是如何从可阅读的文本转变成可被计算机执行的命令的？编写你自己的编译程序又意味着什么？

当你输入或使用外部函数库时，它们是从哪里来的？输入一个外部函数库到底意味着什么？你的编译程序、操作系统或者虚拟机是如何将多个代码段连接起来，形成一个连贯系统的？

掌握这些知识可以使你在技术选择上向"专业人士"跨近一步。

(2) 在工作中或者工作外寻找一个教课的机会。你所传授的知识是自己想要深入学习的技术。在第 2 章第 14 节我们会讲到，讲课是最好的学习方法。

9　切忌孤注一掷

9　切忌孤注一掷

在我负责管理一个应用程序开发团队时，曾问过我的一名雇员："你的职业规划是什么？你将来想要成为什么样的人？"他说："我想成为一名 J2EE 架构师。"这个答案让我十分失望。我问他为什么不想做一名"微软 Word 设计师"或者是一名"RealPlayer 安装者"。

这个人想要把自己的职业道路建立在一门特定的技术上，这门技术是由一家特定的公司创造，而他自己又不是这家公司的雇员。这家公司要是停业了呢？如果这家公司现在热门的技术有一天过时了呢？为什么要把自己的职业发展完全依赖于一家技术公司呢？

不知道为什么，在这个行业中，我们常常欺骗自己说市场的主导和标准是一个概念。所以一些人就认为把其他公司的产品作为自己产品的一部分是合理的。更有甚者，把自己的职业发展建立在非市场领导的产品上。到事业惨败时，他们除了思考自己失败的职业规划，别无选择。

之前我们讨论过，我们应该把自己的职业规划当作是一门生意。尽管我们创立的生意可以寄生于其他生意（比如那些创造移除间谍软件的产品来弥补微软浏览器安全漏洞的公司），但作为个人来讲，这样是十分冒险的。一个公司，就像我刚才提到的创造移除间谍软件的公司，通常可以应对市场的突然变化，比如说微软突然改进了浏览器安全的缺陷（或者说微软决定要进入移除间谍软件这个市场），但是作为个人来讲，通常没有足够的盈余资金来突然改变自己的职业方向或者职业重心。

供应商的软件实施细节是秘密的，导致以特定技术厂商为中心的观点不能成立。你对某

一个软件了解得再多，也会遇到专业服务障碍。专业服务障碍是由该软件公司人为创造的，在你无法解决某些问题的时候，这个公司就会向你出售他们的支持服务了。有时候这类障碍是故意设立的，有时候是那个公司为了维护产品知识产权（不透露源代码）的副效应。

因此，尽管一心一意地投资在一项特定技术上不是明智的选择，但是如果你必须这么做，那么别选择商业性质的，考虑一下开源的。即使你不想或者不能在工作中利用开源方法，那就把开源作为一个平台，使自己可以对一项技术进行深入学习。例如，你想成为了解 J2EE 应用程序服务器是如何工作的专家，那你要做的不是去致力于研究如何配置和部署一个商业应用程序服务器的细节（毕竟，任何人都会在 config 文件中调整设置，对吗？），你应该去下载一个开源 JBoss 或者 Geronimo 服务器，留出时间来学习这些服务器内部是如何运作的，而不是只学习如何操作。

很快，你就会发现你的观点已经自然而然地转变了。这个 J2EE（或者任何你选择深入学习的的技术）也没什么特别的。现在你已经了解了实施的细节，也知道了工作中有高水平的概念模式。你开始认识到，无论是使用 Java 还是使用其他编程语言或者是平台，分发企业体系结构就是分发企业体系结构。你的视野被拓宽了，你的思想也开放了。比起那些特定厂商的技术，你会发现经过你大脑分类解析的概念和模式更易于扩展，也更能被广泛应用。我要说："任那些厂商来去自由吧--我知道如何设计一个系统。"

练　习

试着做一个小项目，做两次。第一次尝试使用在家里就能使用的技术；第二次，使用你最惯用的竞争性技术。

10　热爱它，不然就离开它

10　热爱它，不然就离开它

这听起来像是拉拉队长在大喊加油，目的是刺激你进入一种理想化的疯狂状态。如果你想在工作中做出成绩，就必须对工作充满激情；如果你不在乎这份工作，那后果也会显现出来。

我和妻子刚刚搬到印度班加罗尔的时候，我特别兴奋。在我的职业道路上，第一次期待寻找与我志同道合、对学习充满热情的技术专家。我很期待下班后活跃的生活，我们聚在一起深刻地讨论软件开发方法和技术。我期待看到印度的硅谷被高手们挤得爆满，他们都充满激情地来到这里，寻找软件开发的最高境界。

但是，我看到的大多数人都是来寻找一份工作，很少有人能称得上是热情的开发人员。

这种情景和我的家乡一样。

当然，那个时候我还没有意识到那里和纽约一样。在美国我有一些数据点，但是我一直认为自己在一个不怎样的城市里的一个差劲的公司氛围中工作。我认为这种境况就像我作为一个门外汉第一次踏入 IT 这行时一样。大多数软件开发师都能明白我的意思--我只是没有找到适合自己的工作环境。

通过朋友 Walter 的推荐，我开始在大学的 IT 学院工作。Walter 经常见我用电脑工作，他认为比起大学里那些需要帮助的 IT 工作者，我更能胜任这份工作。可我自己并不这么认为，毕竟自己没受过正式培训。那时，我就是一个爱玩电脑游戏的萨克斯风手。但是，Walter 帮我填了申请表，还安排了面试。面试中基本上没提及一个与技术相关的问题，就这样，我被雇用了，而且即刻上岗。

刚开始工作的时候，我总是疑神疑鬼担心别人发现我是个江湖骗子。我怕别人说"这个萨克斯风手混在这些接受过培训的专业人士中干嘛？"毕竟，与我一起工作的人都是有高级计算机技术学位的人。而我，拿着个音乐专业的学位，却在这行里滥竽充数。

几周后，真相渐渐浮出水面。那些和我一起工作的人--拥有计算机技术硕士学位的人，他们根本不知道自己在做什么！事实上，有些人居然还在看我如何工作，然后做笔记！

知道真相后，我的第一反应就是假装周围都是傻瓜。毕竟，我没有参加过专业的培训。晚上我在酒吧乐队里演奏，白天沉溺在电脑游戏里。我是因为感兴趣才学习怎么用电脑工作。其实，我学习编程就是因为想自己设计电脑游戏。晚上从喧闹的酒吧回到家里，我还可以用编程软件浏览 Gopher 直到天亮。然后上床睡觉，再起床，接着学习，直到不得不出门去演奏。我的生活就是研究我热爱的电脑游戏，吃饭，然后回到 Gopher 或者任何能让我开始工作的编译程序。

现在回头看看，那时候我是入了迷了，但这是一件好事。我的创作欲望被点燃了，这和我开始创作古典音乐或者即兴演奏爵士乐的时候差不多。任何我能学到的东西，都让我痴迷。我这么做并不是为了开始一个新的职业，事实上，我在音乐这行的朋友认为我这样是不务正业。但我的痴迷，令我无法自拔。

这就是我和我那些受到过高等教育，但工作表现却欠佳的同事之间的不同--热情。

这些人不知道自己为什么在 IT 这行工作。他们偶然进了这行，是因为他们认为做编程收入不错，是因为他们的父母鼓励他们，或者是因为他们上大学时想不到什么更好的专业。不幸的是，他们的工作表现将这一切都揭露得一清二楚。

想想你读过的人物传记或者看过的那些关于伟人的纪录片，虽然这些人都身处不同的领域，但是他们都有一个共同点--痴迷，热情。据报道，伟大的爵士萨克斯风手 John Coltrane 练习非常刻苦，甚至练习到连嘴唇都破皮流血了。

当然，在工作能力上，天赋占了很大的比例。不是每个人都能成为莫扎特或者 Coltrane。

但是，我们大可以通过找到自己热爱的工作来摆脱平庸。

一门技术或者一个商业领域可能会使你感到兴奋；相反，或许是某一特定技术或者商业领域拖累了你。也许你更适合一个小团队或者大团队，而你处在不合适自己的组织里；或者是在挑选偏呆板和偏灵活的程序上出了差错。不管是什么原因，想想自己到底适合什么。

短时间内你可能可以伪装，但是缺少热情总会影响你自己和你的工作。

练　习

(1) 找一份自己真正有激情去做的工作。

(2) 下星期一开始，做个简单的日志，坚持两个星期。每个工作日起床的时候，给你的兴奋度打分，分值最高 10 分，最低 1 分。1 分代表你宁愿得病也不想去上班，10 分代表一想到马上就要开始新一天的工作了，你就兴奋，不能再躺在床上 1 分钟了。

两个星期后，检查这个日志。图表中有峰值吗？走向是怎样的？这些点都处在高点还是低点？如果这是一份考卷，那你的平均分是多少？

接下来的两周，每天清晨计划如何在明天得到 10 分。思考你今天要做什么，以便使明天成为你迫不及待要开始工作的一天。每天记录下前一天的兴奋值。如果两周后，这个图表显示的结果还是不尽人意，那或许是时候考虑做一次大的改变了。

做一名机会主义者

做一名机会主义者

--程序员和摄影师

James Duncan Davidson

本文的一开始，我就要告诉读者，我从没考虑过传统的职业规划，我的职业道路是由一个又一个机遇连贯而成的。我的第一个机遇出现在大学时期，我的专业是建筑。十五六岁的时候，我就决定将来要成为一名建筑师，为此我投入了很多。但是上学的时候我痴迷于 BBS，这也为我毕业后的职业选择埋下了种子。我喜欢家里 PC 机上 300 波特的调制解调器，它把我引领到了 Internet，进而让我接触到了 Gopher 和万维网。

我一下就被万维网诱惑住了。我一个接一个地建立个人网站，利用手边的任何技术，有需要的话就自学。那时候，我把这些当作是网络架构的实验。现在听起来这一想法有些好高骛远，甚至有些傻，但这就是万维网刚出现时我们的生活，我们在想象未来将会是什么样子。

当然，Internet 的未来不是在实验室里建造的，它发生在世界经济中。很快，有一家新成立的公司找到了我，他们在为像希尔顿和商业服务管理监督局这样的单位制作网站。他们

看了我的公共网站，很显然我的技术正是他们所需要的。这份工作的薪水在当时来看高得有些离谱。我想我就大干一番，存点钱，几年之后还可以再回到学校。这样，我接受了这份工作。

那年是 1995 年。我当时并不知道事情的发展会有多快，也不知道自己愿意学点什么样的新东西。

希尔顿酒店网站的第一版中有实时预定的布局，在帮助建造这个网站的时候，我学到了如何使用各种各样的服务器来建造网站。我从学徒做起，几个月后，已经可以创造我自己的服务器结构了。回头看看，这一切似乎有些荒谬。但在那时，这是必需的。我看到了一个机遇，抓住了它，最大程度地利用它，然后按照需要重新改造自己。

这件事引发了下面的故事。1997 年，我到 JavaSoft 去做服务器软件。几年后，我在这个公司的工作以负责 Servlet 规范为结束。但很遗憾，这是个投资不足的项目，而且没有一个团队能帮助我工作，比如建立一个新的参考实现。但我并没有因此而停止，我开始从头建立起一个新的实现，这就是后来发行的 JavaServer 网络开发工具包。可能很多人现在已经不记得这个软件了，但是大多数在服务器上使用 Java 的人都应该知道接下来发行的 Tomcat。这个软件是通过 Apache 软件基金会和它的一个老搭档 Ant 发行的。发行背后的故事可以写成一本书了。我可以满意地说，这一切都是因为我能够最大程度地利用每一个机遇。

除了从一名学建筑的学生变成了一名计算机技术人员之外，我曾经还是一名摄影师。祖母教了我摄影的基本知识，父母也鼓励我。所以从我记事起，我就随身带着一个照相机。这曾是我生活中很重要的一部分。在离开 Sun 公司后，我为自己编写了一些软件，这些软件都没有发行，它们都是我用来处理照片的。

2005 年，在我从一名建筑系的学生成为一名软件开发师 10 年后，一个朋友给我打电话，他在 O'Reilly 会议集团工作，他们需要一个摄影师，问我有没有兴趣。我答应了。但是我做的远不止简单拍几张照片。我发狂般地工作，参与每个重要会议，向 Flickr 上传照片来提供最及时的更新。我再次被他们聘请，四年中围绕着它我拓展了客户群，创建了自己的业务。

写这篇文章的时候，我还在时不时地编程，也为一些客户做点软件方面的工作。但是，近期我工作的重点是摄影，我几乎成了一名专业的摄影师。或许有一天，这一状况会改变，毕竟，谁也无法预测未来。

但有一点我可以肯定，我是一个机会主义者。当某事引起了我的兴趣，让我感动兴奋，我会立刻奔向它，不惜一切努力使之成功。通常这都需要学习新的技术，开发新的潜能。有人会觉得学习新东西是缓慢费力的过程，但是我喜欢学习新事物。毕竟，新技术可以让你展开新的工作。我永远不会用自己的技术来定位自己，而是用我已经做过的和我将要做的事情来定位自己。技术只是一种做事的方法。

11　学习钓鱼

第 2 章 在产品上投资

作为一名萨克斯手，我颇具天赋，我可不是在吹牛，先来听我解释我为什么这么说。我做全职萨克斯手的时候，演出很多。忙的时候，一天就会有两到三场。一天内，我会在早午餐的时候演奏爵士乐，婚礼晚宴的时候演奏舞曲，然后在聚会或者在酒吧里演奏 R&B。我之所以说自己有天赋，是因为我发现自己在工作中不断学习，并且不断进步。我对音调很敏感，光听就能学习新的曲子，还能即兴创作。

但是作为一名萨克斯手，我确实没花什么心思。什么事情都是唾手可得，我觉得自己也很满足。在乐队里我是典型的关键人物，所以我的同伴也没让我感到过有压力。

年轻的时候我进步很快，我没有发现我在慢慢地停滞不前，但是随着演奏 R&B 次数的增多，我演奏的曲子听上去越来越相似。每个晚上我演奏的曲调都如出一辙。我即兴演奏的独奏曲也都是重复前晚或者以前演出时演奏过的曲调。现在想想，我觉得不只是我一个人这样，我周围的专业音乐氛围就是这样。我们从不挑战自己，观众也没有让我们感受到挑战。你见过观众因为萨克斯风手持续吹奏一个音超过 30 秒而鼓掌欢呼吗？

最近这些年来，我一直让自己忙忙碌碌，这样就无暇顾及音乐。很长一段时间我完全没有碰过我的萨克斯风和吉他。直到最近，我才认识到音乐在我生活中的重要性，认真地重新拾起了我的萨克斯风和吉他。这次，我在当地没有音乐界的朋友，没有时间全职演奏，由于生疏，演奏得也不再那么出色了。这次，我只是为自己而演奏。

我不敢确定是不是因为自己更成熟了，也或许是更有智慧了，这次我发现只是花一点小小的心思就起了很大的作用。我并没有拿出乐器就开始演奏，这次我必须自己独自演奏。我听音乐然后记录下想要学习的技巧。比如，我想做到 Phil Woods 在高音萨克斯风独奏时的每一个细节；我还想学习 Prince 在 Purple Rain 专辑中的 "Let's Go Grazy"这首曲子中是怎么让吉他发出那样尖锐的声音的。

事实证明，在天赋的帮助下，我只要投资几个小时就能使"这些我总想要做的事情"成为现实。而且随着我投入的时间增多，能力也随之提高。学会一个技巧就有助于学会下一个，攻克一个障碍练习部分，就推动我想要攻克下一个。

就这样专注练习了几个月后，我演奏得比以往任何时候都出色，甚至胜过我做职业萨克斯风手的时候。我在自己的能力（或者说是兴趣）上的投资，彻底击败了那个认为一切都是因为天赋的我。

这就是一个强有力的证据。如果你想要拥有一份可以在职场上出售的产品，一份让你具有竞争力的与众不同的产品，你就必须要在这个产品上投资。在商业中，有想法，有天赋的

人很多。只有向这件产品中投入心血、汗水、眼泪和资金，才能使它真正具有价值。

在本章中，我们将会就职业投资策略展开讨论，探讨如何选择某种技巧和技术来投资，以及不同的投资方法。本章，工作将真正开始。

11 学习钓鱼

老子曾说："授人以鱼，不如授人以渔。"这句话说得非常好。但是老子没有提到如果这个人不想学习怎么钓鱼，第二天又向你要另一条鱼怎么办。有老师也要有学生才能构成教育，但是大部分人都不愿意成为学生。

那么在软件界，"鱼"是什么呢？它是一个过程，一个你使用某种工具、某种技术的某一个方面，或者获取你工作领域的某一特定信息的过程，它是如何从你团队的源控制系统中找到一个特定的子目录，或者是启动一个应用服务器用于开发。大多数人认为这些细节都不是什么问题。你可能会想，会有人来处理这些问题的。源代码管理系统出现问题了，那就去找创建它的人来处理。基础架构小组知道怎么在你和客户之间建立防火墙，所以出现问题时，就给他们发封邮件让他们处理。

谁会希望自己总是任人摆布？如果你想要雇佣某人来为你工作，你会希望这个人总是要受那些专家的支配吗？我不愿意。我想要的是一个能够自立的员工。

很明显，你的出发点应该是学习如何使用你所处行业的工具。以源代码管理为例，它是一个非常强大的工具，它的功能中很重要的一点就是使开发人员的工作更具效率。你不应该把它视为你放置代码的地点。它是你整个开发过程中的重要组成部分。"工作权威资料库"很重要，但别让它束缚你。一个独立的开发人员如果检测过一个项目，那么在存储器中同时存在最后和最好的两个版本的时候，他很容易就能辨别出这两个版本之间的区别。或者你需要对最终发行的代码进行漏洞修复。深夜里你突然间发现一个致命的系统漏洞，你总不会想打电话向别人求助，找到正确的版本然后再开始解决问题吧。这种问题存在于 IDE、操作系统，以及任何一个基础环节中。

你使用的技术平台也是同样的重要。打个比方，你正在使用 J2EE 开发应用系统。你知道应该设置不同的类别，端口和部署描述文件，但你知道为什么要这么做么？你知道这些设置是如何被使用的么？当你启动一个 J2EE 容器时，到底发生了什么？你可能不是一名应用服务器开发人员，但是了解这些可以帮助你为一个平台编写安全代码，当出现问题时，也可以很快地解决问题。

一种最简单的懒办法就是使用向导为你生成代码，这在 Windows 开发中尤其盛行，这些开发工具使很多工作变得非常简单。但是同时，它也使很多 Windows 开发员根本不知道向导背后的代码是如何工作的，向导的工作永远是一个迷。别误会我的意思--正确的代码生成工具是一种非常有用的工具。例如，代码生成器可以将高级 C#代码翻译成能在.NET 运行

的字节代码。你当然不愿意自己编写这些字节代码。但是，站在更高的层次上看，使用向导会让你的知识浅薄，你永远也不懂得向导做的工作。

或许你会很容易就忽视掉这行的"鱼"。如果你在一家抵押贷款公司工作，在估算利率的时候，你可以向专家请教，或者你可以自己学习怎么估算。但是与客户的互动是非常重要的，清楚地了解客户的要求比似懂非懂然后自己填写细节要好得多。如果你真正了解你所工作的行业的详情，那会大大提高你的工作效率。你不需要懂得每个行业的每一细则，但你至少应该了解最基础的规则。与我合作过的出色的软件业人士，很多人都成了客户所处行业的专家，甚至比他们的客户更了解那个行业，这样他们的工作成果显然更好。忽视行业性质的人，往往会犯低级错误，只要懂得最基本的行业知识，这些错误是完全可以避免的。而且，与那些了解行业知识的开发人员相比，这些人的工作效率要低得多（最终导致成本的增加）。

对我们软件开发人员来说，老子所讲的道理，或许可以翻译成"要一鱼，食一日；要人授以渔，终身受用。"但是，请注意，不要要求别人来教你，要自己主动学习。

练　习

(1) 如何与为什么？--在你读书或者工作的时候，想一想工作中你不完全懂的问题。你可以问自己这两个问题：它是如何工作的？为什么会发生这种情况？

对这两个问题，你可能给不出答案，但是只要你问了，就会形成一种新的思维模式，也使你更加关注自己的工作环境。IIS 服务器是如何通过向 ASP.NET 发送请求来结束工作的？为什么我必须要为我的 EJB 应用程序生成这些接口和部署描述文件？我的编译程序如何处理动态和静态链接？如果店主住在蒙大纳，为什么计税的方法就不同了？

当然，这些问题的答案很有可能会引发对这些问题的新一轮探索。当在这个"如何和为什么"的环节中你无法再深入了，那就证明你已经达到目的了。

(2)"提示"时间--在你的工具箱里挑选一种非常重要却经常被忽视的工具。可能是你的版本控制系统，可能是一个你广泛使用却只知皮毛的库，也可能是你用来编程的编辑器。

选定了工具后，每天花一点时间学习这项工具的新知识，帮助你提高工作效率，或者能让你更好地掌控开发环境。比如，你可以选择操作 GNU Bourne Again Shell (bash)。当你的思绪游离出手头的工作时，你可以上网查询关于使用 bash 的提示，而不是装载 Slashdot。很快，你就可以找到有用的资源来教你如何使用 shell。现在，有了新的诀窍，你就可以利用一系列"如何和为什么"的问题来深入研究它的核心了。

12　学习行业是如何运转的

12　学习行业是如何运转的

在上一章，我们讨论了仔细选择业务领域的重要性。不能小看业务领域的知识，它可以决定雇用方是否会选中你，而且也会让你在工作中赢来阵阵掌声。开始学习某一行业的细则之前，应该确定所做的选择是适合自己并且适合市场现状的。

但是，有一种知识既不属于技术范畴也不是特定于某一行业的，而且也不会很快就过时，它就是财务基本知识。不管你在哪一个行业工作--制造业、医疗部门、公益机构或者教育系统都是一个行业。行业本身就是一门你必须要学习的知识。

我记得当我还是一个年轻的程序员时，参加一个员工会议。我呆滞地盯着一位公司的高管，他向我们展示着一组又一组的数据图表。我永远也不会直接与这位高管共事，他所展示的数据与我完全没有关系。我低声嘀咕着"我就想回到办公桌前，完成我手头上的应用程序功能。"我的团队成员们坐在一起，就像是长途旅行车上局促不安的孩子。我们没人关心会议内容，根本不知道为什么叫我们来参加这个会议。我们责怪这个无能的经理召开这个会议，浪费我们的时间。

现在回想起来，我才知道我们当时是多么无知。我们来这个公司工作，目的就是为它赚钱或者省钱，但我们根本就不懂这行是怎么赚钱的。更糟糕的是，我们根本就不认为这是我们应该知道的知识。作为程序员和系统管理员，我们认为自己正在做的工作就是我们应该做的。但是，如果连这行是怎么赚钱的都不知道，又怎么能创造性地帮助公司赚取利润呢？

上一段中有一个词--创造性地--是关键所在。没错，我们是 IT 专家，这也是公司付钱雇我们的原因。有了合适的项目和领导班子，我们就应该努力做这个业务，根本不需要了解这行是怎么运作的就能为它提供价值。这些想法看似有道理。

但是，有创造性地增加价值需要全面地了解你所工作的行业环境。在商业世界，我们常常听到"账本底线"这个词。但到底有多少人真正理解"账本底线"是什么，以及什么能对它产生作用？更重要的是，又有多少人知道自己怎么做才能对这个"账本底线"起到有利的作用呢？你的组织是赔本还是盈利（你是为其创造了利润还是给它造成了损失）？

了解你公司的财务运作可以让你做出有意义的转变，而不是茫然无知地专注于某一事情，却主观地认为这样就是对的。

练　习

(1) 通读一本基础商业教程，一本 MBA 教程是不错的选择。我推荐一本非常有用的书 The Ten-Day MBA[Sil99]。你真的可以在 10 天内读完，占用不了多少时间。

(2) 找一个人带你到公司的财务部门看看，并请他们向你讲解财务状况（如果你的公司不介意与员工分享这些信息）。

(3) 听完财务状况讲解后，再向他们复述。

(4) 弄明白为什么"账本底线"要被称为"账本底线"。

13 寻找良师

13 寻找良师

爵士音乐文化中很重要的一部分就是寻找良师。在爵士乐这行，年轻的乐手从师于一名有经验的乐手是很普遍的。这些有经验的乐手收他们为徒，传授他们爵士乐的真谛。不止如此，师傅们除了教授知识外，还会成为职业顾问和生活顾问，年轻的乐手会向他们征求意见。同时，年轻的乐手对师傅非常忠实，会围绕着他们的良师建造一个狂热的乐迷网络。

这样乐手与乐手之间就有了关联，而乐手也通过这样的关系网找到了工作机会。爵士乐文化是一种围绕师徒关系的自组织文化和习俗。这个系统起到了很好的效果，甚至让人怀疑这是由某个组织机构领导的。

在传统的职场世界里（特别是 IT 业），我们很少能求助于他人。依赖别人被看作是脆弱的象征。我们害怕承认自己不够完美。竞争无处不在，只有强大的人才能在竞争中生存下来。遗憾的是，这种观念导致了"师徒机制"严重不发达。如果你去问爵士乐手："你的师父是谁？"他们每个人都会有一个答案。但是如果向程序员提出同样的问题，在美国，他们的反应会是："什么？"

以前情况不是这样的。西方历史中职业教育的繁盛可以追溯到中世纪。比起音乐领域的人，技术工作者受到的专业培训更强大也更正式。年轻人在开始职业生涯的时候，都会做学徒，从师于某个有名望的技术工作者。他们得到的工资很少，但换来的却是向这名大师学习的特权，而这名大师的任务就是把学徒训练成像他自己一样技艺高超的人。

一名良师最首要也是最重要的任务就是做一个榜样。直到亲眼见识到某人突破你所熟悉的极限时，你才知道什么才是一切皆有可能。榜样的作用就是定义何为"好"。作为一名棋手，战胜家里人可能会让你感觉不错，但是如果你和一名参加大赛的选手较量，你就会知道原来下棋是一项如此深奥的游戏；当你和一名大师下棋时，你又会得到另一个启示。如果你只是一直和你家里人下棋并且战胜他们，你可能会觉得自己是个高手；没有榜样，就没有动力进步。

良师还可以将你的学习过程形成体系。在上一章中，我们说过选择在哪种技术和行业领域中投资时，你会有很多的选择。有时候，太多的选择会让你不知所措。按常理说，前进总比静止不动要好得多。良师可以帮助你削减这些选择，避免你白费精力。

我刚开始做一名系统支持的时候，我紧紧抓住一个叫 Ken 的人，它是我们学校网络管理员之一。一次我们学校的 NetWare 网络系统出了问题，想要进入学校数字图书馆的学生的电脑都会受到危害，Ken 帮我解决了这个大难题。在那之后，他就再没给过我什么启发（他

也没有试图给）。当我让他为我指明怎样才能更有见识，更加自立的时候，他给了我一个很简单的答案：潜心钻研目录服务功能，习惯 UNIX 变体，掌握一种脚本语言。

在无数的技术中，他为我挑选了 3 种。这个被我视为权威的人，非常自信地为我指明了道路，我开始按照他的建议学习这 3 种技术。我的职业道路就是建立在这 3 种技术的基础之上的，直到现在我做的事情都与这 3 种技术相关。这倒不是说 Ken 为我指明的方向是绝对正确的--毕竟天下没有绝对正确的答案。重要的是他缩小了要掌握的技术范围，这样我就可以学习技术，取得进步，而不是停止不前。

良师还是值得信任的朋友，他们可以观察并判断你做出的决定和取得的进步。如果你是一名程序员，你可以把你的代码拿给他们看询问他们的意见。如果你计划在公司或者本地用户组会议上做一次演示，你可以先在良师面前做一次演示，从他们那里得到回馈。良师是值得你信赖的人，你甚至可以问他们："作为职业程序员来说，我与其他人有什么不同？"你知道他们不仅会帮你分析，还会帮助你取得进步。

最后，就像在爵士乐这个行业里一样，你不仅建立了对你良师的依赖和责任，反过来也一样。当我帮助某人时，我就是在这个人的成功上投资。在他的职业道路上，我向我认为对的方向轻轻地推了他一把；如果这个人沿着这条路走取得了成功，那这也是我的成功。

这样就激励了老师帮助他的学生取得成功。通常，经验丰富的成功人士都会受到一些重要人物的尊重。这个良师就成了你与这个人际关系网之间的桥梁。这种桥梁作用是不能被小看的。毕竟，有句老话是"有本事不如认对人。"

这种师徒关系的形式不重要。不需要特别清楚地要求某人成为自己的老师（当然你要这么做也不是坏事）。事实上，你的导师可能并不知道他在扮演着这个角色。重要的是你要有可以信赖钦佩的人，他可以帮助你做出职业导向，帮助你磨练技术。

练　习

指导自己--我们都希望有人主动来教我们，但事实是我们很难在自己周围找到这么个人。所以要学会自己做自己的良师。

想想在你工作的领域中你最钦佩谁。大都数人在不同阶段都会有一个名单。这个人可能是工作中的同事，或者你很欣赏这个人做出的某项成果。列出这个榜样的 10 种特性，这些特性必须都是视他为榜样的理由。这些特性可能是某一特定的技术方面，比如技术广度或者某一特定领域的知识深度。或者是他们的某种人格魅力，比如他们是很好的团队协调者，或者他们的言辞总是具有吸引力。

现在，把这些特征按重要性的升序排列，1 是最不重要的，10 是最重要的。这样你就提炼出了一个特征列表，这些特征都是你钦佩并认为重要的。这就是赶上你选择的榜样的方法。但是，要先专注于哪一项呢？

想象你的榜样在此项上会给你打多少分（10 分最高），在这个名单上加一列，将分数标注在特征旁边。尽量站在你所选择的榜样的角度上，以第三者的身份来评定自己。

一切准备就绪后，再在整个列表的最后再加一列以记录你的最终得分。如果你给某种特性打 10 分，即最重要的特性，然后对这项你自己的表现给出的分值是 3 分，那这项你的最终分数就是 7 分。得出各项的最终分数后，按降序排列，这样你就能分出改进的先后顺序了。

从这个列表中的前两三项开始，列出你要改进的具体事项。

14　做一名良师

14　做一名良师

要想真正学点东西，可以试试向别人传授这些知识。清楚知道自己是否对某一知识真正理解的最好的办法就是把你的理解讲给别人听，让他们明白。这个简单的方法是公认的帮助你理清思绪的灵丹妙药。在软件开发这行，大家都知道软件开发师经常会对着宠物或者什么无生命的物体讲述如何解决一个问题。

Marin Fowler 曾经在班加罗尔的一次开发师讲座上说，当他想要真正学懂一些知识的时候，就把它们写出来。Marin Fowler 是著名的软件开发师和作家。如果我们把他当成是一名作家，远程传授知识，那他就是这个行业中最负盛名、最有影响力的老师。

我们通过传授知识学习。这听起来有些有悖常理，因为我们希望老师对他要传授的知识了解得一清二楚。当然，我不是在说我们通过向别人讲授的同时就能学到新的真理--要是这样的话，那这些真理是从哪来的？但是，知道真理并不意味着同样知晓它们的前因后果。这种深层次的理解才是可以通过教授学到的。当要阐明复杂问题的时候，我们会用简单一些的事例打比方。我们会弄明白为什么一个类比看起来似乎是可行的，但其实是行不通的，而另一个类比看起来不行，却是行得通的。当你向别人讲授的时候，你就必须回答这些你可能从未想过的问题。通过讲授，我们的那些知识死角就会暴露出来。

所以，就像你找到了一个好的导师，做别人的导师也会使你受益不浅。

做别人的导师也会产生积极的社交效果。一组重叠的导师和学生创造出一个紧密且紧密的社会关系网。在职业关系网中，比起那些熟人来说，导师和学生的关系是最紧密的。当你处于这一关系中时，你们彼此就是忠诚的。在这种关系网中，可以很好地解决难题或者是寻找工作。

你不应该低估帮助别人的感觉--那感觉棒极了！如果你能随时想着别人的利益，那这就是你在用你的技术来帮助别人。当今的经济环境很不稳定，帮助别人这项工作是不会使你下岗的，而且这份工作带给你的收入是不会随着通货膨胀而贬值的。

寻找学生的方法不是你声称自己是权威，而是使自己具备真才实学并且有耐心愿意与别人分享你的知识。如果你并不是某方面的绝对权威，也不要惊慌。有时候你只需要具备某方面的经验，然后去帮助那些比你经验少的人。想想看自己有没有这样的机会去帮助别人。

比如，你可能已经做过大量的 PHP 工作，你就可以参加当地 PHP 用户组会议，向那些比你经验少的使用者提供帮助，帮他们解决具体的问题。或者，你目前还没有这种面对面的辅导机会，很简单，你可以在网络留言簿或者聊天系统中回答问题。但是要记住，这种通过网络建立的师生关系远远比不上面对面的师生关系。

你无需去建立一个正式的师生关系，就从帮助别人开始，好处会自然而然随之而来。

练　习

(1) 找一个你可以帮助的人。你可以在公司找一个比自己资历浅经验少的人，比如一名实习生。你也可以去当地大学的计算机信息技术学院去做志愿者，辅导一名大学生。

(2) 找一个网络论坛，并挑选一个主题。开始帮助别人。慢慢地，你就会因为愿意助人以及有能力帮助别人学习而在这个论坛里出名。

15　练习，练习，再练习

15　练习，练习，再练习

当我还是一名学习音乐的学生时，我经常彻夜在音乐学院教学楼里练习。训练室的墙壁很薄，我演奏的声音经常被一些十分难听的演奏声吞没。这并不是说我们学校的乐手都很差，正好相反，他们在练习。

当你在练习的时候，演奏出来音乐或许总是难听的。如果你在练习的时候，总能演奏出悦耳的音符，那就证明你一直无法突破自己的极限。这就是练习的意义所在。运动也是一个道理。运动员在训练的时候总是将自己推到极限处，这样他们才能在比赛中突破自己的极限。他们让丑陋的东西都暴露在平时的练习中--而不是真正的比赛中。

在计算机这行，经常会有开发师突破自己取得进步。但是，很多时候都是因为他们本来就不胜任自己正在做的工作。我们这个行业习惯于在工作中练习。你能想象一名专业的乐手，站在舞台上，演奏出来的却是大学训练室里那些难以入耳的声音吗？这肯定是让人难以忍受的。音乐家是通过公开表演而赚取报酬--是表演，而不是练习。同样，如果功夫高手或者拳击手在比赛中表现得疲惫不堪，那他在这项运动中也没什么发展。

我们应该寻找时间练习。在西方，与外包给那些国外团队的工作相比，我们经常把相对高水平的编程工作交给当地的开发人员。我们要在质量上与他们做竞争，就不能把工作当作练习来对待。我们要在提高技艺上面投资。

几年前，效仿练习演奏，我开始试着进行编程练习。第一条规则就是练习开发的东西绝对不能是我想要使用的。我不想图方便，仓促地达到目的。所以我开始编写我用不着的程序。

我没有走捷径，但是在练习中我发现很多想法都不能实现，这让我很泄气。尽管我尽可能地把它当做工作来好好做，那些设计和编码却不能达到我的期望。

现在回想起来，当时那种窘迫的感觉是个好的迹象。我编写的代码并不是完全没有亮点。我在开发大脑，突破自己的编程极限。就像练习吹奏萨克斯风时，如果练习的时候演奏出来的都是悦耳的音乐，那我知道我根本没达到练习的目的。同样，如果练习时编出来的程序都是很棒的，那我就是在发挥我的正常水平，而没有接近我的极限--好的练习应该让我接近自己的极限。

那么，我们怎样才能知道要练习什么呢？怎样才能达到极限呢？作为一名软件开发师，就如何练习这个话题，可以单独出本书来讨论。作为开始，我还是会借鉴我作为爵士乐手的经验。我把爵士乐演奏练习分解成以下几个类别（由于很多人不是乐手，所以我把它们简化了）：

身体与协调

视奏

即兴演奏

这个框架可以成为软件开发师练习的一种方法。

身体与协调：乐手要进行乐器演奏技巧训练：发声、身体协调（比如练习手指的灵巧度）、速度和精准度。这些都是非常重要的练习项目。

那对于我们软件开发师来说，这些基础练习又是什么呢？那些初级编程语言中，有没有你基本不怎么使用的？比如，你选择的编程语言支持正则表达吗？在很多的编程环境中，正则表达式非常强大但却很少被使用。大多数开发员即使可以用到它，也不去使用，因为他们的技术没达到那个水平。结果，创造出了很多不必要的字符串解析编码，并且不得不延续使用。

这条规则也同样适用于 API 和库。乐手们常说，对于技术，你要是没达到手到擒来的地步，那它们真能帮到你的时候你也想不起来它们。这就需要尝试深入研究，比如，在你选择的编程环境中，多线程编程是如何工作的。或者 stream 库、网络编程 API，甚至是一切可用的处理集合和列表的工具。大部分现代编程语言都提供了丰富和强大的库，但是软件开发师们只学习了其中的一小部分子集。如果他们掌握了如何使用一整套工具，那么编程的效率就会提高很多。

视奏：对一名专业录音乐手来说，首要能力就是能够在第一时间完美地视奏。我曾经为

Blockbuster 公司（音像租售公司）的圣诞乐曲专辑中吹奏萨克斯风。在一曲大型乐队演奏的快节奏的乐曲中，我演奏了序曲和第二段高音部分。当磁带开始滚动的时候，我才第一次看见乐谱。我演奏了一遍序曲，又演奏了一遍第二部分。任何失误都会使整个乐团重新再演奏一遍，这样就会浪费时间，并且增加工作室的租金。

作为软件开发师，视奏又是什么呢？是需求规格，还是设计？开源社区是找到用来练习的代码的绝佳场所。有没有哪个开源软件是你最喜欢的？你可以给它加个功能。挑选一个你想要用来练习的软件，浏览它的待办事项，给你自己规定时间来实现这个新的功能（或者至少决定要实现这个功能需要哪些步骤）。

选择好功能之后，下载源代码然后开始开发。怎么知道要看哪里？有什么好方法在一组重要的代码中理出头绪？又要从哪里开始呢？

你可以经常进行这种练习，而且这种练习的周期不会持续很长。你并不一定要去实现这个功能，就把它当做是一个开始，练习真正的目的是以最快速度读懂你正在看的代码。但是一定要确保这个软件与你平时工作时使用的软件不同。要寻找不同风格、不同编程语言的软件进行练习。记录下是如何使整个过程增加或者降低难度。你使用了哪些方法帮助你理解这些代码？面对复杂的函数层次，你是以什么为线索，让调用栈有迹可循，带领自己穿梭其间呢？

即兴创作：即兴创作就是在某种结构或者限制的基础上创造新的东西。作为程序员，我发现自己往往在压力之下可以即兴创作。"哦，不！无线网络应用服务器出问题了，我们收不到命令了！"这情景常常发生在最具创造力、最即兴的编程过程中，比如通过无线网络从二进制日志文件中手动重发数据包来修复丢失的数据。没有人特意来为你做这些工作，特别是在这么紧急的关头。这种优秀并且迅速的编程能力在正确的关头发挥作用是非常重要的。

训练思维敏捷和提高即兴编码技术的好方法是通过自我限制的方式来练习。选择一个简单的程序，试着来限制你的编程过程。我最喜欢做的就是编写一个程序来显示那首老掉牙的歌曲"99 Bottles of Beer on the Wall"的歌词。如何能编出一个程序而不做任何变量赋值？或者在保证正确显示歌词的前提下，这个程序最小能做到多小？再加一个限制，你最快用多久能编出这个程序？可以使用定时器，练习编写一个难却小的程序。

这只是一种极限视角的练习方法。你可以从任何学科找到练习的对象，从视觉艺术到僧侣信仰。最重要的是找到你所需要的来进行练习，并且确保你不是在工作中练习。你必须要找出时间来练习，这是你的责任。

练　习

(1) Topcoder--Topcoder.com 是一个很早就存在的编程竞技网站。你可以注册然后通过线上竞赛赢得奖励。就算你对竞争没兴趣，Topcoder 还为你提供了一个练习室，里面有很多可

以练习的问题。现在就去注册尝试一下吧。

(2) Code Kata--Dave Thomas 是《程序员修炼之道》的作者之一，他接受编程练习这个想法，并使之实用化。他创造了一系列的很小却深具启发性的练习，被称之为 Code Kata，程序员可以使用他们选择的编程语言来做这些练习。每一个 Kata 都针对某一特定技术或者思考过程，这样程序员的思维就可以更加灵活。

这本书的印刷过程中，Dave 已经创造出了 21 个 Kata，你可以在他的博客上免费使用（http://codekata.pragprog.com/）。在这个博客上，你还可以看到通讯名单的链接，以及别人解决这个问题的方法和相关讨论。

你的挑战：练习这 21 个 Kata，并撰写使用 Kata 练习的日记（或者是博客）。当你完成这 21 个 Kata 后，开发你自己的 Kata，并与别人分享。

16　做事的方法

16　做事的方法

"软件开发"不是一个名词，而是一个动词词组，它是一个创作的过程。当我们编码的时候，开发的步骤与开发的成果一样重要。如果你忽视开发的步骤，那就有可能会误工，创作出的产品有某种缺陷，或者什么也创作不出来。这些后果都会引起用户的不满。

幸运的是，人们在开发一些优质的软件的过程中（以及其他产品）投入了很多心思。这些现有的技术被编录成"方法论"。你可以在网络上或者书店里找到大量这方面的书籍。

遗憾的是，大部分软件开发师并没有从这些优秀的资源中受益。对一个团队中的大多数成员来说，步骤是事后要想的问题。在他们的字典里，"方法论"这个词等同于报告和冗长的会议。很多时候，某种方法是经理强加给他们的。

经理直觉上知道他们需要遵循某种步骤，但却不清楚他们有哪些选择。结果，他们依然延续 20 世纪 80 年代的办事程序，只不过在其中添加了符合年代特征的时髦用语（现在是 Agile 软件公司），然后经理再把这个办事程序传递给他的团队。当团队里的软件开发师自己成为经理之后，还是会重复这个相同的步骤，直到有人打破这个循环，试着证明哪些可行哪些不可行为止。

你肯定会想一定有某种更好的软件开发方法。事实上对于大多数团队来说，的确有更好的方法。

如果你是一名程序员、测试员或者是软件设计师，你可能会认为开发的步骤不是你的工作内容。就你的公司而言，你或许是正确的。但是，它通常不是任何人的责任。如果非要把这个工作分配给某个人，那可能就得单成立个"步骤小组"或者类似的某个不相连的机构。事

实上，一个成功的软件开发步骤，必须是由使用它的人来参与制定的--类似你这种人。

要想让自己找到对这个步骤的归属感，最好的方法就是亲自来操作。如果你的机构中没有工作步骤，那研究方法论对你可能会有帮助。吃午餐的时候，和你的同事讨论开发软件中会碰到的问题，以及选择某种规范步骤或许会缓和这些问题。把你们选择的步骤整合成一个计划，然后得到每个团队成员的认同，并开始执行你的计划。

你工作的环境中，步骤也有可能是由上级传达下来的。当这个文件到达开发团队，可能已经变了味，或者被重新解释成了新的模样。像中国的一个游戏"耳边传话"，要传递的话从第一个人传到最后一个人那里时，可能会完全走了模样。那这就是一个开始行动的机会。研究这个方法，试着向你的团队和管理部门说明它最初的意思。你无法反对这个上级下达的步骤，那不妨通过正确的阐述使之运作起来。

方法论听起来像某种时髦的名词，很空洞。但是，这对软件开发步骤的研究会有所帮助--即使是研究你并不需要去做的事情。如果你很擅长软件开发过程，那你就拥有了一个更有力的论证来证明你的团队如何才能更好地工作。

即使有很多角度的方法论可供选择，一个公司也不可能完全照搬某个方法。没关系，一个能使你的团队工作更有效率，帮助你们生产出更好的产品的步骤就是最好的选择。

方法论：不只是给电脑发烧友的

尽管项目管理不一定与软件开发方法相关，但你可能会发现自己已经开始接触公司的项目管理技巧。在这个行业里，人们使用着大量的项目管理方法。其中最著名的要属项目管理学院所出的"Project Management Book of Knowledge①"了（本书以其著名的认证制度闻名）。

另一个非针对软件的高质量方法论是六西格玛②。由 General Electric 和 Motorola 这样的公司引领，六西格玛方法强调测量、对过程的分析以及生产效率和用户的满意度。六西格玛严谨的系统方法直接适用于程序员的日常工作。

① http://www.pmi.org/。

② http://www.isixsigma.com/。

唯一找到项目管理与软件管理混合方法的办法是研究可用的选择，挑选出适用于你和你团队的方法，从真正的实践中不断提炼总结。

如果你不能沿着这个程序走，那你就很难生产出产品。比起找到一个做软件开发的人来说，想要找到一个能够设计出软件开发步骤的人要难得多。所以，学习软件开发步骤的知识可以祝你一臂之力。

练　习

选择一个软件开发方法论，并且挑选一本有关此方法论的书，登陆相关网站，加入一个讨论此问题的联络组。从批判的角度来研究此方法论。此方法论的优势和弱点在哪里？在你的工作中，执行它的障碍是什么？研究完一个，再换一个继续研究。对比他们的优势和弱点，想一想如何能把它们结合起来。

17　站在巨人的肩膀上

17　站在巨人的肩膀上

作为爵士乐手，我会花大量的时间听音乐。我听音乐，不只是在阅读或者开车的时候把它当做背景音乐，而是真正在聆听。如果爵士即兴创作就是演奏你听到的琴弦拨动出的乐曲，那么认真听音乐是一种非常重要的灵感源泉，也可以帮助你判断什么是可行的，什么是不可行的；什么是出色的，什么是一般的。

爵士唱片的丰富历史是一套惊人的知识体系，供相关人士聆听。因此，对爵士乐手来说，听音乐不是一种被动的行为，而是一种学习。更重要的是，能够理解你所听的音乐是一种日积月累培养出的能力。我音乐圈里的朋友确实在做这种听音乐的练习。我们会举办这种听音乐的聚会，在聚会上爵士乐手聚在一起听音乐然后讨论。有时候我们其中一个人会放一张即兴独奏唱片，其他人根据这个风格判断出这个即兴演奏的人是谁。

当然，爵士乐手没什么特别的。古典音乐作曲家也这么做，小说家和诗人，雕刻家和画家同样如此。研究大师的作品是成为大师的一个重要步骤。

听爵士唱片的时候，我们会讨论即兴独奏乐手传递音乐主旨的技巧。"哇！你能听出来末尾处他是用什么方式回旋的吗？"或者，"在第三个八拍中，他滞后于节拍的演奏方法真奇怪。"通过这样的讨论，我们发现这些技巧，并吸收精华，下次即兴表演时可以拿来试用。

从这方面讲，软件设计和编程与音乐是相同的。我们可以从大量的现有程序中寻找模式和技巧。设计模式运动（见 Design Patterns [GHJV95]）关注可重复使用解决软件开发问题的方法。设计模式使现存程序的研究正式化，使大量专业软件工程师得以进行这项工作。但设计模式也只是我们读程序编码的时候可供使用的知识中的一个小子集。

那么其他程序员是如何系统地解决某一特定问题呢？其他人是如何有策略地使用变量、函数和结构命名的？如果想在一种新的语言中执行 Jabber 即时信息协议，该如何做？在处理 UNIX 和 Windows 系统的进程间通信时，又能有什么创新的方法呢？通过学习现有的程序，这些问题都可以迎刃而解。

比找出某一特定问题的解决方法更重要的是，将现有的程序当做一面放大镜来检查你自己的编程风格和能力。就像每当我听 John Coltrane 的唱片时，都会提醒自己作为一名萨克斯手，我站在技术阶梯的哪一层，欣赏编程大师的作品也会起到同样的作用。尽管如此，这

不仅仅是说我们自己要谦卑。当你读这些程序的时候，你会发现某些你可能永远也不会去实践的工作，甚至是你想都没想过的。他为什么要这样做？他是怎么想的？他这样做的目的又是什么呢？即使你读到的是不怎么样的程序，通过批判的角度来研究它，你也会有所收获。

这种向已出版的作品学习的行为在艺术界更加容易，因为一幅画或者一首乐曲的背后并没有隐藏的源代码。你听到了一首曲子或者看到了一件艺术品，那你就能从中学习。谢天谢地，我们软件开发师可以通过开源软件来研究无数可使用的现存软件。

开源软件数量很多，所以要想全都研究完也是不可能的。当然这其中肯定存在一些不好的开源项目，但也有不少是非常优秀的。有些开源编码甚至可以在任何编程语言中完成任意软件能够完成的任何任务。

当你以一种批判的视角去看这些程序的时候，你就会开始培养自己的品味，就像你对音乐、艺术和文学的品味一样。不同的风格和技巧可能会使你觉得好笑、惊讶、气愤，或者像我所希望的，让你觉得有挑战。如果你是真的在认真学习它们，在设计范例碰到问题的时候，你会更具创造力。就像在艺术这行，你学习别人的习惯时，就会发展出自己独特的风格。

阅读这些程序的另一个作用就是能让你知道哪些方法是已经存在的。当你碰到一个待解决的问题时，你可能会记起曾经在处理这样或者那样的项目时，看到一个执行 MIME 类型的库。如果这个方法是正确的，那你就会因为使用已经存在的方法而节省了很多时间，同时也为你的公司节省了开支。当你意识到在软件这个行业中，正因为我们一遍又一遍地重复发明轮子（发明这个词用在这里太笼统了）而浪费了多少金钱时，你会大吃一惊。

牛顿说过："我看得更远，是因为我站在巨人的肩膀上。"像牛顿一样的智者都清楚地知道我们能从先人身上学到很多东西。做一个像牛顿一样的人。

练　习

(1) 选择一个项目，像读书一样研读并且做笔记。归纳出好的方面和坏的方面。发表一篇评论。至少找到一个你可以借鉴的技巧或者模式。再找到至少一处缺点，提醒自己在开发软件的时候不要犯这类错误。

(2) 找到一些志同道合的人，每个月聚会一次。每次聚会由一个人提出一段代码 --2 行或者 200 行都可以。分解它，然后讨论这段代码背后的东西。思考做出这个程序所做的决定，权衡没有包含在这个程序中的代码。

18　在工作中，将自己自动化

18　在工作中，将自己自动化

作为 IT 界的管理人士，我期望把项目承包给低成本（国外）的咨询公司，但同时又坚

信雇佣最便宜的开发师通常并不能把项目的成本降到最低。这个斗争在我的职业生涯中一直没有停止过。我与 IT 总监或者副主管进行过很多次激烈的争论，我强烈要求雇佣一些真正优秀的开发师而不是雇佣一大批成本低但技术水平也低的写代码的人。

遗憾的是，每次我都是话还没说完就得闭嘴。并不是因为我的观点有问题（显而易见没问题!），而是我很难证明自己是正确的。从成本来看，能够证明他们的观点是正确的唯一证据，就是较低的按小时计算的开销确实有利于公司节约雇佣成本。

想象一个你能想到的任何软件开发项目。如果这个项目要在三个月内完成，那需要雇用多少程序员？5 个还是 6 个？（别嫌我烦）如果同样一个项目要在两个月内完成呢？如何节省出一个月的时间？

IT 部门主管的标准答案是--要想加快项目进程，那就增加程序员的数量。这是不对的，但是大家都这么认为。如果你可以通过增加人手来加快一个项目的进程，这样推断的话，也就是说越多的人就意味着更高的工作效率。

要想达到相同目标，其实有很多种方法。如果目标是提高软件开发的效率，你可以：

找到工作效率更高的人来做这个项目

找更多的人来做这个项目

自动化工作

由于目前为止我们还无法真正衡量软件开发的生产率，因而也就很难证明一个人比另一个人工作的速度快。所以财务经理关注的是一小时支付的薪酬高低。这就引出了一个简单的公式，适用于一个规定的时间段：

$$生产率 = \frac{项目数量}{程序员数量 \times 平均时薪}$$

在某些情况下，也许可以确切地计算出软件项目的投资收益。但多数情况下，怎么计算项目个数，怎么统计需求数量，都没有既定、统一的标准，而且这些标准依情况不同无法简单地套用。

所以雇佣工作效率高的程序员这个方法是很难证明对错的，我们也不会鼓励增加廉价程序员这种方法。那么剩下的方法就只有自动化了。

还记得 20 世纪 80 年代美国失业率极高的时候，我们不仅责备外来打工的人，还责怪机器，特别是计算机。工厂里都安装了大型的臂状机器，他们比人类产量高而且比人类精准，

这样人类与他们好像根本就没什么可比性。每个人都非常低落，除了这些机器人的发明者。

把你的公司想象成一家为小公司建立网站的公司。基本上你所做的就是一遍又一遍地创立相同的网站，网站上有联系方式、产品概述、购物车以及相关事物。你可以雇一小部分工作效率高的程序员创建这个网站，然后雇一组廉价的程序员一遍又一遍地手动重复所有工作，或者你可以创建一个系统来自动生成这个网站。

我们向财务经理的计算公式中插入一些（虚构的）数字，就得到了图 2-1 中的等式。

自动化属于我们这行的 DNA。但是某些原因致使我们不自动化我们的工作。你怎么能够有理有据地比外包团队更加迅速和廉价地开发出更好的软件呢？制造机器人，把你自己变为自动化。

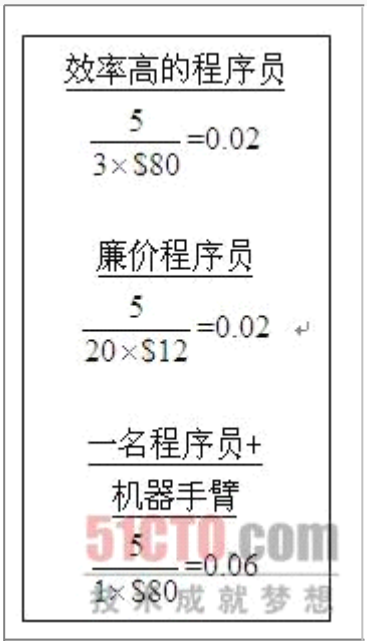

图 2-1　生产率对比

练　习

(1) 挑选一个你经常重复做的工作，为它编写一个代码生成器。从简单的部分做起。不要管这个代码生成器的重复使用率，只确保这个代码生成器可以节省你的时间。

想办法提高生成代码的抽象等级。

(2) 研究模型驱动架构（MDA）。尝试一些可以使用的工具。看看工作中哪里可以使用 MDA 概念。想想如何用你日常使用的工具来应用 MDA 概念。

从 IT 顾问到常务董事

--Enterprise Corp 常务董事 Vik Chadha

我从通用公司的 IT 顾问，成为 bCatalyst（拥有 5 百万基金的企业加速器）公司的一名创业合伙人，这并不是我预先设想好的职业规划。

那我又是怎样离开一个拥有上万名员工的《财富》排名前五强的企业，到一个刚刚开始创业并且在一项还不成熟的高科技项目投资的公司中工作的呢？现在回头看看，这过程中出现了一些零散的点，我把这些点连接起来，发现了一些重要的图形，在这里我想和大家分享，希望你们可以有所借鉴。

在弗吉尼亚理工大学取得电子信息工程硕士学位后，我加入了 GE 公司，成为了一名 IT 顾问。因特网的商业用途开始盛行，我接手了几个项目，这些项目都是为这个强大无比的平台和它的技术支持所设计的，我与公司的每个团队合作--从 IT 财务团队到技术服务团队，再到销售人力自动化团队，最后到销售数据仓库团队。我喜欢研究最新技术，然后运用它们来解决商业难题。

但是，研究尖端技术也不总是有趣的。我们在研究这些还不成熟的技术过程中，常常碰壁，花费了大量的时间和精力为使用者调试产品。从客户的角度看，不管一种技术看上去多么出色，只有当它能够解决真正紧急的难题并且让他们真正受益时，它才算得上出色。久而久之，这一观点使我从以技术为中心的思维方式向以解决方案为中心的思维方式转变。几年之后在 bCatalyst 公司评估新技术启动的时候，这种新的思维方式充分体现了它的价值。

尽管我在通用工作得非常开心，但总觉得还是少了点什么。作为 IT 专业人士，我觉得自己一直以来都只是在一个方面发展自己的技术，没有机会学习公司到底是怎么运作的，利润是怎么产生的，又是怎样继续积累的。我没有因此而气馁，我决定行动起来，更多地去了解商业知识，以及学习如何做一名企业家。我没有参加过任何商业课程的学习，要想学习这里面的来龙去脉，唯一的方法就是实践，反复试验，从失败中学习。

我以前一个想要创业的室友，也是我的好朋友 Raj Hajela，我妻子 Vidya，还有我，我们三个人集思广益，试图发掘出现在市场的需求。我们想从电子商务中寻找商机，但却不想销售任何产品。我们有一些艺术相关的背景知识，并且对艺术都非常感兴趣，因为每一件艺术品本质上都是独一无二的。我叔叔就是一名艺术工作者，以此养家糊口，只是生活非常艰辛。通过调查研究，我们得出结论--大部分艺术家的生活都是这样清贫。所以，我们决定要创建一个平台来解决这个问题。通过这个平台，帮助艺术家展示推销他们的作品，并与他们的资助人保持联系。这样，我们就发布了 Passion4Art.com，开始寻找艺术家加入我们的网站，把他们的电子作品放到网站上来展示。当我们与 1 000 名艺术家签订合约，并且制作了他们自己的网站后，我们开始相信自己所做的工作是有意义的，并且开始寻找资金。

那时候（大概是 1999 年），一家叫做 eMazing.com 的公司就不同的话题提供每日小贴士。我们想或许我们可以和他们合作（我们提供艺术家，他们提供分销渠道）提供每日艺术小贴

士。他们公司的一名高管与我们会面，赞成我们的观点，并决定试行。

我们告诉他为了建造基础结构，我们正在寻找注资，他提出可以把我们的商业企划案寄给城里的一家新企业加速器公

司--bCatalyst。

几天后，我们接到 bCatalyst 执行总裁 Keith Williams 的电话，他想要与我们面对面地谈谈，进一步了解我们的项目，我们相当地兴奋。我后来才知道他们是通过一个可靠的消息提供者对我们进行了深入地了解，直到那时我才知道这有多重要。所以，当你试图得到风险投资商的青睐时，尽力找到一个热心的推荐人，因为这是得到面谈机会的最好办法。

通过和 Keith 的几次交谈，我们发现这次合作是可行的。不过当时网络泡沫开始出现，在这上面投资对他们来说不是个好时机。在最后一次会面上，他们表明非常喜欢我们的团队，但是这不足以让他们出资。如果我们可以再想出一个吸引他们的点子，他们就会毫不犹豫地支持我们。我问他们是否真想与我们合作，还是礼貌地拒绝我们。他们向我们保证，言出必行。

后来我又约 Keith 见面，告诉他我愿意从通用辞职，在接下来的几个月全职与他们工作，共同寻找其他的机会。这样一来，他们既不必做出长期的承诺（类似在你买下一个程序前，我们提供试用的机会），也降低了我们的风险。我说服他们相信我愿意离开通用，而且是在没有后路的情况下离开，这就说明了我的决心，说明我愿意全身心地投入到这个项目中。

接下来的一年里，每天我们都会与不同的团队见面，他们向我们推销自己的点子试图得到我们的支持，我注意到我们向每个公司提问时出现了一套新模式。

我把这套提问汇编成了一个列表并与你分享，希望以后你需要风险投资商赞助的时候能用得上。（详见 http://www.enterprise- corp.com/resources/assessment.htm）。

那一年里我在 bCatalyst 公司学到的技能成就了今天我在 Enterprise Corp 公司常务董事的职位。过去 7 年中，我与 100 多个公司合作过，帮助他们募集到 7 500 多万的资金。如果那时候我没有冒险尝试新事物，就无法得到这么丰富的经验。这条路上的蜿蜒曲折是不可或缺的部分。读者朋友们，我希望我的故事可以激励你们找到自己独特的道路--一条可以充分发挥你能力的道路。